Fall 12-15-2022

ON-BOARD ARTIFICIAL INTELLIGENCE FOR FAILURE DETECTION AND SAFE TRAJECTORY GENERATION

Eduardo Morillo
Embry-Riddle Aeronautical University, morilloe@my.erau.edu

ON-BOARD ARTIFICIAL INTELLIGENCE FOR FAILURE DETECTION AND SAFE TRAJECTORY GENERATION

By

Eduardo Xavier Morillo Guerra

A Thesis Submitted to the Faculty of Embry-Riddle Aeronautical University

In Partial Fulfillment of the Requirements for the Degree of

Master of Science in Aerospace Engineering

November 2022

Embry-Riddle Aeronautical University

Daytona Beach, Florida

ON-BOARD ARTIFICIAL INTELLIGENCE FOR FAILURE DETECTION AND SAFE TRAJECTORY GENERATION

By

Eduardo Xavier Morillo Guerra

This Thesis was prepared under the direction of the candidate's Thesis Committee Chair, Dr. Hever Moncayo, Department of Aerospace Engineering, and has been approved by the members of the Thesis Committee. It was submitted to the Office of the Senior Vice President for Academic Affairs and Provost, and was accepted in the partial fulfillment of the requirements for the Degree of Master of Science in Aerospace Engineering.

[30pt]

THESIS COMMITTEE

Graduate Program Coordinator & Chairman, Dr. Hever Moncayo	Date
Member, Dr. Richard Prazenica	Date
Member, Dr. Maj Mirmirani	Date
Dean of the College of Engineering, Dr. James W. Gregory	Date
Associate Provost of Academic Support, Dr. Christopher Grant	Date

This work is dedicated to my parents Edgar and Paulina, my sister Michelle, my grandparents Edgar, Anita, Rosita and Guillermo, and my aunts and uncles. Your pure and unconditional love has made this possible. Thank you from all my heart.

ACKNOWLEDGMENTS

First of all, I want to thank God for giving me a shower of blessings throughout my professional and personal life which has lead me to have the necessary strength and wisdom to complete this work. He has put on my path people among family, friends, and colleagues whose heart truly express the significance of love and friendship.

I am extremely grateful to my parents, Edgar Morillo and Paulina Guerra, for their love, sacrifice, and caring to educate and prepare me for my future endeavors. I am thankful to my sister, Michelle Morillo, my aunts Grace Morillo, Doris Morillo, Maritza Guerra, and Alina Ruiz for their unconditional support, as well as my uncle, Mauricio Guerra, who has given me courage in hard times. Also, I am utterly grateful to my grandparents whose sacrifices and hard work throughout the years have lead to the creation of our beautiful family.

This study would not have been possible without the guidance and support of Dr. Hever Moncayo, for which I am eternally grateful for giving me the opportunity to be part of the Advanced Dynamics and Control Laboratory at ERAU. I could not have asked for a better advisor. In addition, I want to thank Dr. Richard Prazenica for playing a key role in my formation as an Aerospace Engineer, as his wisdom during the lectures and great human qualities were the main source of inspiration for choosing the Dynamics and Control concentration track. I also want to express my gratitude to Dr. Maj Mirmirani, Dr. Pam Daniels, and the Aerospace Department Faculty for their help throughout the years.

I want to thank my chosen family: Andrei Cuenca; Juan A. Leon; Dennis Moreno; Silvana Ureña; Patricia Velasco; Juan F. Granizo; Lorraine Acevedo; Santiago Restrepo; Juan Pava; Derek and Abbigail Espinosa; Yogesh Pai; Yash Meta; Anish Prasad; Rocío Jado; Gabriela and Gustavo Gavilánez; Christoph Aoun; Ayush Raminedi; María José Jacome; Pedro Reina; Sebastián Espinosa; Alexander Cisneros; Yaser Ronquillo; David Cando; Pedro Sandoval; and all of my friends who have been there for me. There are no words to describe how grateful I am of having you in my life. I am thankful for your love, care, and support in the hard times and the good times. This work is also dedicated to you.

ABSTRACT

The use of autonomous flight vehicles has recently increased due to their versatility and capability of carrying out different type of missions in a wide range of flight conditions. Adequate commanded trajectory generation and modification, as well as high-performance trajectory tracking control laws have been an essential focus of researchers given that integration into the National Air Space (NAS) is becoming a primary need. However, the operational safety of these systems can be easily affected if abnormal flight conditions are present, thereby compromising the nominal bounds of design of the system's flight envelop and trajectory following. This thesis focuses on investigating methodologies for modeling, prediction, and protection of autonomous vehicle trajectories under normal and abnormal flight conditions. An Artificial Immune System (AIS) framework is implemented for fault detection and identification in combination with the multi-goal Rapidly-Exploring Random Tree (RRT*) path planning algorithm to generate safe trajectories based on a reduced flight envelope. A high-fidelity model of a fixed-wing unmanned aerial vehicle is used to demonstrate the capabilities of the approach by timely generating safe trajectories as an alternative to original paths, while integrating 3D occupancy maps to simulate obstacle avoidance within an urban environment.

TABLE OF CONTENTS

LIST OF FIGURES

Figure **Page**

LIST OF TABLES

NOMENCLATURE

ABBREVIATIONS

ABC	Aircraft-Body-Centered Reference Frame
AC	Abnormal Condition
ADCL	Advanced Dynamics and Control Lab
ADIS	Adaptive Immune System
AIS	Artificial Immune System
BIS	Biological Immune System
CFD	Computational Fluid Dynamics
DCM	Direction Cosine Matrix
DR	Detection Rate
EOMs	Equations of Motion
ERAU	Embry-Riddle Aeronautical University
FA	False Alarm
FFC	Formation Flight Control
FGC	Formation Geometry Calculation
HMS	Health Monitoring System
INIS	Innate Immune System
NAD	Number of Activated Detectors
NAS	National Air Space

NED	North-East-Down Reference Frame
NLDI	Nonlinear Dynamic Inversion
NS	Negative Selection
OBC	On-board computer
PI	Performance Index
PS	Positive Selection
RRG	Rapidly-Exploring Random Graphs
RRT	Rapidly-Exploring Random Tree
RRT*	Rapidly-Exploring Random Tree-Optimized
RSDUM	Raw Data Set Union Method
SLAM	Simultaneous Location and Mapping
sUAS	Small Unmanned Aircraft Systems
SVM	Support Vector Machine
UAS	Unmanned Aircraft Systems
UAV	Unmanned Aerial Vehicle
VTOLs	Vertical Take-off and Landing Vehicles

SYMBOLS

α	Angle of Attack (AoA)
$\bar{\gamma}$	Limit of the Flight Path Angle
$\bar{c}$	Mean Aerodynamic Chord
β	Side-Slip angle
$\boldsymbol{z_e}$	Final Pose
$\boldsymbol{z_s}$	Initial Pose
χ_{free}	Obstacle-Free Space
χ_{obs}	Obstacle Space
$\Delta\delta_{ail}$	Relative Deflection of the Ailerons
δ_T	Desired Thrust
δ_{ail_L}	Left Aileron Deflection
δ_{ail_R}	Right Aileron Deflection
δ_{rud_L}	Left Rudder Deflection
δ_{rud_R}	Right Rudder Deflection
$\dot{\psi}$	Rate of Change of Heading
$\dot{r}_e$	East Velocity
$\dot{r}_n$	North Velocity
ϵ	Voxel Size
γ^*	Optimal Flight Path Angle

γ^c	Commanded Flight Path Angle
Γ_i	Constraints of the System
$\gamma_{nominal}$	Nominal conditions of Flight Path Angle
$\mathfrak{E}$	Envelope Relevant Variables
$\mathfrak{F}$	Set of Features
$\mu(\chi_{free})$	Volume of Obstacle-Free Space
ϕ	Roll Angle
ϕ^c	Commanded Roll angle
$\phi_{nominal}$	Nominal conditions of Roll Angle
ψ	Heading angle
σ	Feasible Path
θ	Pitch Angle
$\tilde{C}$	Accumulated Actuation Energy
$\tilde{E}$	Accumulated Error of Attitude Tracking
$\tilde{T}$	Accumulated Error of Translational Displacement Tracking
ζ_d	Volume of the Unit Ball
b	Wing Span
c	Wing Chord
d	Dimension of the State Space
E	Set of Edges

f	Position Forward Error
I_{xx}	Moment of Inertia along X-axis
I_{yy}	Moment of Inertia along Y-axis
I_{zz}	Moment of Inertia along Z-axis
k	Number of Turns of Each Helix
l	Position Lateral Error
m	Aircraft Mass
mem_{grid}	Memory Size in Bytes of 3D Grid
n	Number of Nodes or Samples
$N_{feat_{ail}}$	Number of Features for aileron failure cases
$N_{proj_{ail}}$	Number of Projections for aileron failure cases
p	Roll Rate
$Pb/2V$	Helix Angle or Normalized Roll Rate
PI_{global}	Global Performance Index
r	Yaw Rate
R_B^E	DCM for Body-to-Earth Coordinate Transformation
R_{min}	Minimum Turning Radius
r_{res}	Map Resolution in Meter/Cell
S	Wing Planform Area
V	True Airspeed

v_δ	Directly Involved Variables
v_ϵ	Equivalent Directly Involved Variables
V_n	Vertex Set
W	Aircraft Weight
x, y, z	Inertial Reference Frame Positions
x_{goal}	Goal Region
x_{init}	Initial Condition
$x_{nearest}$	Nearest Node
Y	Set of Features at Post-failure Condition
z_{de}	Down-Component of the Final Pose in the NED Frame
z_{ds}	Down-Component of the Start Pose in the NED Frame

1 Introduction

The use of unmanned aerial systems (UASs) has been exponentially growing during the past decade, especially due to their versatility, relatively low-cost manufacturing and operational flexibility. These vehicles are capable of carrying out different type of missions that range from different applications including recovery operations, disaster reliefs, construction inspection and law enforcement applications among others. Consequently, increased levels of autonomy are becoming a main focus of researchers since potential failures in the systems and/or subsystems of the UAS can represent serious harm to civilians and high risk of damage to private and public property.

Trends in the increase of registration of recreational sUAS and commercial UAS have been exponentially growing in the last decades, and it continues to grow as more commercial and recreational applications are developed for these type of vehicles. Currently, the FAA [1] forecasts that the recreational UAS fleet will increase from 1.32 million units to around 1.48 million units by 2024, while the commercial UAS are expected to grow to approximately 828,000 aircraft for the same year. The fact that UAS are becoming more popular demands that their integration to the National Air Space (NAS) has to be safe as well, meaning that an increase in autonomy capabilities is required. Nevertheless, abnormal flight conditions due to mechanical, electrical, and software failures have been imposing a great barrier in several UAS applications, especially where the flight environment is challenging. Loss of power, control surfaces lock, battery explosions or even malfunction of the on-board computers can cause a UAV to easily injure a person or cause damage in property which can lead to expensive costs and law penalties [2].

As a result, researchers have been focused on improving control law algorithms by using methods such as artificial neural networks for enhancing the stability of the vehicle among others [3],[4]. Some studies even propose the use of other UAS such as air blimps in order to reduce the risk of collision that a malfunction might cause [5]. However, this still does not present a viable solution towards the original problem.

Improved system autonomy requires accurate strategies to detect and predict abnormal flight conditions as well as innovative fault-tolerant control strategies to mitigate the risk of flying in challenging environments [6],[7]. Researchers have tackled this problem by implementing algorithms related to the on-line health monitoring of the system, which is dedicated to the identification of the abnormal flight condition once the vehicle is subjected to a failure [8]. On the other hand, some efforts focus on improving path planning algorithms such that collision avoidance can be achieved with maximum efficiency while taking into account the vehicle's original mission goals. Despite these improvements, there has been a lack of research in terms of safe trajectory generation based on a reduced flight envelope.

1.1 Literature Review

This section presents the preliminary knowledge that has been compiled using available literature that will serve as a basis for presenting the decision-making architecture of this thesis. The main focus relies on providing essential information about three areas of knowledge that reside inside the autonomous guidance and navigation area: Occupancy maps for obstacle avoidance, path planning algorithms, and the AIS paradigm for failure detection, identification, and evaluation.

1.1.1 Occupancy Maps for Obstacle Avoidance

For autonomous navigation in any robotics application, the knowledge of occupied and free space on a 3D environment is essential for collision-free path planning. There are many methods available in the literature to model 3D maps. Ranging from point clouds and elevation maps to multi-level surface maps, each method presents its own advantages depending on the desired application.

The study performed by Newman et al. [9] uses a mapping method based on constructing point clouds for a SLAM (Simultaneous Localization and Mapping) application for modeling a urban environment. Point clouds are usually obtained from laser sensor data and are set of coordinates in the Cartesian plane that are classified according to their elevation and intensity. For this case, the authors used a laser in combination with a vision system to sample the

local geometry of the environment to incrementally build the 3D map. Though this process is implemented in an on-line configuration, it highlights some general characteristics of point cloud maps. As being a method that demands a high detection rate of points in the space, high precision sensors are often used to capture in detail the geometrical shapes of the environment. Though obstacle avoidance improves greatly, memory and computational consumption are a major drawback of this method, therefore leaving it unfeasible for implementation with path planning algorithms.

Furthermore, improvements to these type of classical mapping techniques have been implemented for reducing size memory and allowing path planner algorithms to be simultaneously implemented. The work done by Douillard et al. [10] shows the combination of the points cloud method with the partition of the mapped world into a set of voxels, also known as single-cubic units of volume, such that 2D grids can be extended into 3D objects without a substantial use of memory. Additionally, they build specific hybrid maps where they combine multi-surface elevation maps with voxel creations such that detailed physical characteristics of the terrain can be represented, therefore providing an enhanced model of the environment. This is also closely related to the previous work done with elevation maps, also known as $2\frac{1}{2}$ maps, where each cell on a two-dimensional grid contain the information of the height of the obstacles [11]. According to the authors, these types of renderings are feasible to be implemented with path planning algorithms. However, the main drawback of this study relies on the pre-processing time that is needed to obtain the final map.

Ryde and Hu [12] have presented a novel multi-resolution algorithm that allows to arrange 2D data stored in voxel lists to produce 3D maps. They formally present the volumetric space occupied by the voxels, and they use a 3D probability function of position $p(x, y, z)$ to describe the probability density of the occupancy at a particular position in the state-space. Equation 1.1 shows this, where ϵ is the voxel size, and the indices are defined by Equation 1.2.

$$P_{i,j,k} = \int_{z}^{z+\epsilon} \int_{y}^{y+\epsilon} \int_{x}^{x+\epsilon} p(x,y,z)dx\,dy\,dz \tag{1.1}$$

$$i = [\frac{x}{\epsilon}],\ j = [\frac{y}{\epsilon}],\ k = [\frac{z}{\epsilon}] \tag{1.2}$$

The drawback for this implementation is that the authors do not consider the difference between free and unknown volumes, which can affect the performance of the path planner when trying to establish boundaries on which the state-space can be searched for generating alternative paths once the flight envelope is altered due to an abnormal condition.

From these studies, it can be noticed that several implementations and constructions of 3D mapping are limited by their own properties. For instance, the need for a 3D map that contains the following properties is required:

- Compact enough to be memory efficient and computationally inexpensive.
- Capability of differentiating occupied space, free space, and unknown space.
- Detailed enough to accurately represent volumetric obstacles.
- Capability of having multi-resolution properties for high-level path planners used for navigation.

The 3D mapping framework that meets these requirements is based on the OctoMap, a relatively new architecture that relies on structures called octrees for memory saving and volumetric representation. This framework was firstly proposed by Wurm et al.[13] and has continuously being improved by several researchers [14],[12]. The authors explicitly demonstrate the advantages of their proposed method over the previous mentioned mapping methods in terms of compactness, flexibility, and updateability. Figure 1.1 [13] shows a comparison between point cloud maps, elevation maps, multi-level surface maps and the proposed OctoMap technique as shown by the authors. Notice that the OctoMap structure

provides an improved map of the same volumetric object with higher accuracy that has even a reduced memory size. The memory size in bytes of the 3D grid that stores the information about the occupancy level of the voxels can be calculated using Equation 1.3, where r_{res} is the map resolution measured in meter/cell and x, y, z are the bounding coordinates in each dimension.This 3D map presents a viable solution for autonomous navigation.

$$mem_{grid} = \frac{x \times y \times z}{r_{res}^3} 4B \tag{1.3}$$

Another advantage of this framework is that it has been released under the BSD-license which allows any user to access freely to the source code. In addition, the libraries are made to support Windows, Linux, and Mac OS X with their respective compilers, plus compatibility with Robot Operation System (ROS) is also available for Linux Ubuntu distribution. The authors have well documented the support for these platforms, and these can be find in `https://github.com/OctoMap/octomap` or `https://octomap.github.io/` along with the code repositories. Chapter 2 presents the theoretical details of the OctoMap method and the application that it has for the present work in terms of modeling and simulating fixed obstacle avoidance.

1.1.2 Path Planning Algorithms for Autonomous Vehicles

Nowadays, several type of robots are able to perform autonomous navigation tasks due to the extensive amount of research and studies that engineers and scientist have conducted in the last three decades about path planning algorithms. Ranging from UAVs, UASs, underwater autonomous vehicles, ground vehicles, even to common vacuum cleaners such as the iRobot, this technology has had a great impact on the goal of increasing autonomy for artificial intelligence applications. It would be an exuberant task to address all the current existing path planning algorithms, their details, and their enhanced versions in a single study. However, the most important ones will be mentioned as choosing the optimal algorithm for the present application is extremely important as this framework will have to interact with

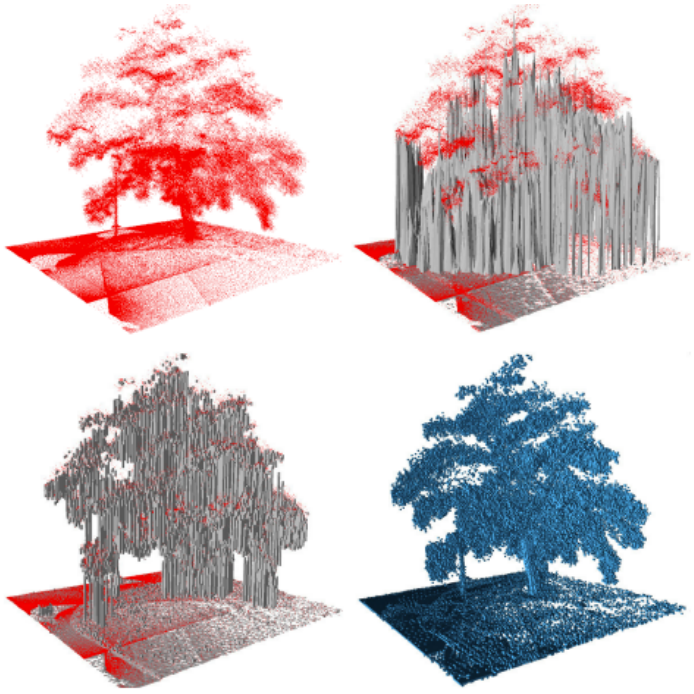


Figure 1.1 Comparison between Point Cloud Map (top left), Elevation Map(top right), Multi-level Surface Map(bottom left), and OctoMap technique (bottom right).

the AIS paradigm.

Path planning is an essential tool that allows almost any type of robotic intelligence to identify safe, efficient, collision-free paths from a start position to a destination [15]. The vast majority of these algorithms have been enhanced mainly to improve computational efficiency, re-planning tasks, and memory saving. Path planners can generally be classified differently according to their characteristics, but an efficient way to differentiate them is according to their own way of generating feasible paths for the vehicle. The work performed by Yang et al.[16] provides a survey of the main categories in which the most popular path planners can be classified. Though most of the literature focuses on 2D algorithms due to their simplified complexity, one the purpose of this thesis is to provide a real-world solution to the safe trajectory generation. Therefore, 3D mission planners will be addressed. These can be classified in five main categories according to the methodology they use to create the trajectories. Figure 1.2 [16] shows the main classification and their most important details are highlighted.

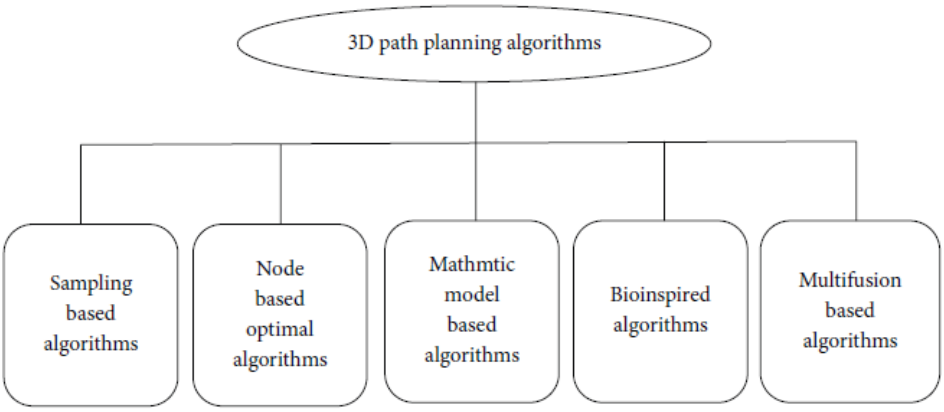


Figure 1.2 3D Path Planning Taxonomy.

Sampling based algorithms: These type of planners require preknown information about the environment (mainly for obstacle avoidance and limit bounds), though they can still sample the nodes or cells in spaces. They can be subdivided into two further categories: active and passive, as shown in Figure 1.3 [16]. The main difference is that the active planners can find an optimal solution based on the definition of an heuristic cost function, while the passive planners do not guarantee an optimal solution even if this one exists. Table 1.1 [15] describes the most common heuristic functions that researchers use to optimize the path planners, with the Euclidean distance being the most popular one. Here, x and y represent Cartesian coordinates.

Table 1.1 Heuristic functions for path planning optimization.

Function	**Equation**
Euclidean distance	$\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$
Manhattan distance	$\|x_1 - x_2\| + \|y_1 - y2\|$
Octile distance	$max\|x_1 - x_2\| + \|y_1 - y2\|$

These planners sample the path as a set of nodes or cells, similar to a Monte Carlo sampling method. In this category, Rapidly-Exploring Random Tree (RRT) [17] is included along with its optimized version (RRT*), which uses the Euclidean heuristic cost function. Active planners also include another variant of the RRTs known as Dynamic Domain RRT (DDRRT),

and the well known Artificial Potential Fields. On the other hand, passive path planners include 3D Voronoi diagrams, Rapidly-Exploring Random Graphs (RRG), and Probabilistic Road Maps (PRMs) along with its variants. More information on these algorithms can be found on References [6],[15],[18].

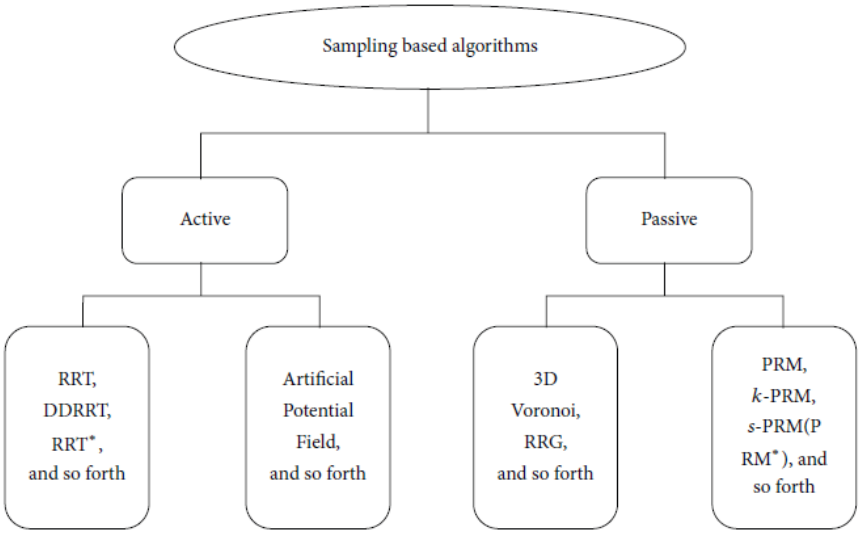


Figure 1.3 Sampling Based Path Planners.

Node based optimal algorithms: These type of algorithms are similar to the sampling based algorithms in that some previous information about the space is known, such as a 3D map once obstacles are previously defined or detected by sensors. The difference relies in that they use a graph decomposition method to plan the path, which usually can also be optimized. Well known algorithms in this category include the Dijkstra's algorithms (A* and D*), Theta*, and Lifelong Planning A* (LPA*) as shown in Figure 1.4 [16]. Dijkstra's [19] algorithms and its variants can also be classified as search algorithms, and they usually deal with static environments. LPA* [20] are more robust as they are able to handle dynamic environments as well as D*-Lite [21]. Regardless of their search method, all these algorithms deal with discrete optimization based on graph decomposition.

Mathematic model based algorithms: These type of algorithms include also sub-categories which include linear algorithms and optimal control. Some of them deal with kinematic constraints that have to do with the environment modeling, and the dynamic

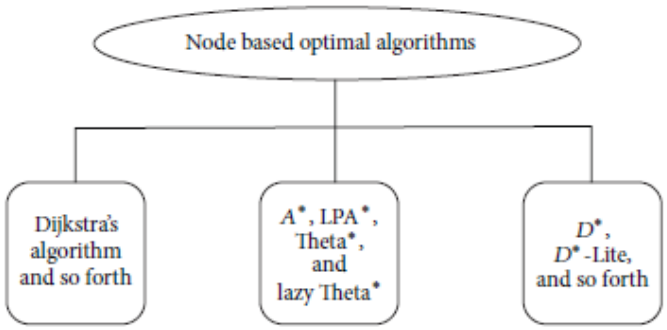


Figure 1.4 Node Based Path Planners.

constraints of the vehicle. In fact, these algorithms contain almost all the information about the kinodynamic constraints present, but at the same time this causes them to require high time complexity and increased computational effort. More on these type of algorithms can be found on References [22],[23],[24].

Bioinspired and multifusion based algorithms: Bioinspired algorithms try to mimic the nature of mainly animals to find feasible paths for the autonomous vehicles. They are subdivided into Neural Network (NN) and Evolutionary algorithms, with the latter ones being widely used to replicate the behaviour of insect colonies [16],[15]. For example, the behaviour of ants, bees, and even bat colonies have been studied which has lead to the creation of the Ant Colony Optimization algorithm (ACO) [25], the Bat Planning (BA) algorithm [26], and the Artificial Bee Colony algorithm (ABC), respectively. Additionally, the concept of nature evolution has been replicated in planners such as the Genetic Algorithm (GA)[27] which has allowed to solve complex problems involving multiobjectives. However, problems in convergence and also high time complexity are the main disadvantages of these planners.

Often, Bioinspired planners are combined with the sampling based algorithms which has lead to multifusion based algorithms. The objective of making a fusion of both types of path planners is primarily to counteract disadvantages that each one presents. A clear example of a multifusion algorithm is presented in the work done by Lin et al. [28] where the BA algorithm is used to counteract the problem of local minima that Artificial Potential Fields present [18].

Given that there is a numerous combination of path planner algorithms and enhancements of each one of them that has been carried out over the years as computational capabilities have improved, is extremely important to choose a convenient path planner for the present application. In fact, the path planner will also have to take into account the physical and dynamical constraints of the vehicle in order to produce a feasible path. Due to this, it is important to define the basic concepts of holonomic and nonholonomic systems.

1.1.3 Holonomic and nonholonomic systems

In the local approach, *nonholonomic systems* are defined as those mechanical/robotic systems in which velocity constraints are not originating from positional constraints [29]. This means that velocity is not integrable into a positional constraint, from which it is impossible that the system returns to its original position due to its prior states (i.e. path taken). On the other hand, *holonomic systems* are those characterized of having integrable constraints into positional constraints. Intuitively, this means that nonholonomic systems cannot move in all directions locally in the configuration space while holonomic systems are able to move freely in all directions [30].

A clear example of a nonholonomic system is a fixed-wing UAV, where the system cannot move freely as its dynamics are constrained by the path it has already taken. Clearly the fixed-wing depends on the previous discrete position that it held so that based on that position, the actuators can be able to make it reach just certain future positions in the next time step due to the geometrical and physical constraints of the system (i.e maximum aileron/elevator/rudder deflections). For instance, the path that this UAV followed puts a constraint into its future discrete positions at every time step.

In contrast, a quad-copter UAV can easily return to a specific position and orientation regardless of its previous states and path taken. Intuitively this is obvious, since the position and orientation are not constraints due to surface actuation. Therefore, the quad-copter is characterized as an holonomic system. Further details are provided in chapter 3.

1.1.4 Choosing RRT* as a path planner:

The on-board capabilities for the architecture that is presented in this work requires real time on-line path planning, for which the selected planner will need a feasible time complexity and computational efficiency. Additionally, a planner that takes into consideration the kinematic constraints (related to the modeled environment), and dynamic constraints of the vehicle will be of primary need (kinodynamic constraints), especially if an abnormal condition is present since this directly affects the dynamical behaviour of the vehicle, for which safe trajectories would be able to be generated under a restricted flight envelop.

Under the time complexity consideration, the on-line real time requirement, and whether the environment is static or dynamic (S/D) in terms of the obstacle avoidance, Table 1.2 [16] presents a classification of the previously mentioned algorithm categories. Here, n is the number of nodes/samples of the algorithm and the "Big O" notation is used to represent the growth function that characterizes the time complexity. Typically, this way of representing algorithms according to their run time and space requirements is used in the computer science field to have an idea of the algorithm's performance.

Table 1.2 Properties of path planning algorithms.

Algorithm category	**Time Complexity**	**S/D Environment**	**Real Time**
Sampling based	$O(n \log n) \leq T \leq O(n^2)$	S and (some) D	On-line
Node based	$O(n \log n) \leq T \leq O(n^2)$	S and (some) D	On-line
Mathematic model based	*Depends*	S and D	Off-line
Bioinspired	$O(n^2) \leq T$	S and (some) D	Off-line
Multifusion	$O(n \log n) \leq T$	*Depends*	Off-line

Notice that sampling and node based algorithms are similar in computation, and they are on-line methods. For instance, the search is reduced to these two categories. If kinodynamic properties are considered, the feasible algorithms for the present study suggest that sampling based methods must be used. This is essentially due to the fact that node based algorithms are not suited for nonholonomic systems, but rather just for holonomic systems.

Further research suggests that in addition to time complexity and kinodynamics being essential requirements, the most efficient path planner will have to present additional characteristics to ensure the creation of feasible safe trajectories. These entire set of characteristics is presented in Table 1.3 and briefly described below.

Table 1.3 Ideal characteristics of the path planner.

Characteristics of the Path Planner
Kinodynamic properties
Probabilistic completeness
Asymptotic optimality
Monotone convergence
Acceptable time and space complexity

Probabilistic completeness has to do with the probability of the path planner to be able to find a solution as the number of samples approaches infinity, while asymptotic optimality is defined as the probability of converging asymptotically to the optimal solution with infinite number of samples [31],[32]. A path planner is probabilistic complete and asymptotically optimal when both probabilities are 1. Moreover, monotone convergence has to do with the ability of the solution to converge towards a certain bound, and space complexity has to do with the amount of memory used by the algorithm to compute the samples. Formal mathematical definitions of these concepts are further presented in chapter 3.

The study conducted by Karaman and Frazzoli [31] presents a complete analysis of some active and passive sample based algorithms based on the properties described above, and it proposes new improved versions of the algorithms that meet such properties. Table 1.4 shows the result of their work and provides expressions for time and space complexity for comparison purposes. Recall that these parameters are expressed in terms of nodes or samples n, and the Big O notation as described previously.

Based on this description, it can be noticed that the least efficient algorithm in time and space complexity is the Simplified Probabilistic Road map (sPRM) as its function

Table 1.4 Main characteristics of sample based path planners.

	Algorithm	Probabilistic Completeness	Asymptotic Optimality	Monotone Convergence	Time Complexity	Space Complexity
Original/Existing Algorithms	PRM	Yes	No	Yes	$O(n \log n)$	$O(n)$
	sPRM	Yes	Yes	Yes	$O(n^2)$	$O(n^2)$
	k-sPRM	Conditional	No	No	$O(n \log n)$	$O(n)$
	RRT	Yes	No	Yes	$O(n \log n)$	$O(n)$
Improved Algorithms by Karaman and Frazzoli [31]	PRM*	Yes	Yes	No	$O(n \log n)$	$O(n \log n)$
	k-PRM*					
	RRG	Yes	Yes	Yes	$O(n \log n)$	$O(n \log n)$
	k-RRG					
	RRT*	Yes	Yes	Yes	$O(n \log n)$	$O(n)$

presents a quadratic growth. Consequently, this path planner is discarded as well as all the PRMs and RRGs variations since they are passive planners. Now, only RRT and RRT* seem to be feasible candidates to be taken into consideration. However, notice that RRT is not asymptotically optimal which is a limitation for on-line path planning. Nevertheless, RRT* does have all the necessary characteristics while maintaining the same time and space complexity with kinodynamic considerations. This is of special importance since usually the modified/improved versions of the original methods tend to come in hand with increased time or space requirements. For instance, the most feasible trajectory planner to be chosen for this application is the RRT* algorithm.

Dubins Paths: Despite RRT* is suitable for the present nonholonomic application, it cannot act alone since the branches of the tree impose sharp turns that in most cases makes the trajectory unfeasible to be followed, as will be shown. Within this line of thought, it is convenient to smooth the path generated by the path planner such that a feasible path can be created at all time steps. To achieve this, the Dubins paths are considered as firstly proposed in 1957 by L.E. Dubins[33]. Ones the RRT* finds the optimal path between the initial and

final pose, then the Dubins Path is applied into each segment of the trajectory with the main goal of transforming sharp-edge turns into smooth arcs, enabling the fixed-wing to be able to follow the 3D trajectory.

Chapter 3 will present the theoretical definitions of the path planner algorithm and details about the Dubins Path implementation that will also have embedded decision-making capabilities to find the best cost-optimum 3D trajectory.

1.1.5 The Artificial Immune System Paradigm

In recent years, there has been an increased interest in mimicking biological systems through the use of immunological computation techniques to solve a wide variety of problems that include optimization, classification, clustering, anomaly detection, machine learning, adaptive control, and associative memories [34]. Under these categories, artificial immune systems have been developed and continuously enhanced to act similarly to the Biological Immune System (BIS) as a mechanism to recognize and remember specific pathogens within a host organism. Under this idea, Takahashi and Yamada [35] have originally proposed a mathematical representation of the artificial immune system commonly subscribed under what is known as the Artificial Immune System paradigm.

Furthermore, the presented idea has been adapted and improved to be used as a detection, identification, and evaluation mechanism for abnormal flight conditions in complex systems such as aircraft and spacecraft, where failures have been studied at the software and hardware level [36],[37]. In order to understand how the AIS paradigm works, it is necessary to have a basic knowledge of how the BIS operates in the human body.

Biological Immune System: The human body has mainly two subsystems that compose the entire BIS: The Innate Immune System (INIS) and the Adaptive Immune System (ADIS). These are shown in Figure 1.5 [38], and are explained as follows.

Innate Immune System: The INIS is formed by the natural killers of the host organism and present the first line of defense against intruders (antigens). The INIS is formed mainly by white blood cells such as macrophages, dendric cells, neutrophils, eosinophils, and basophils

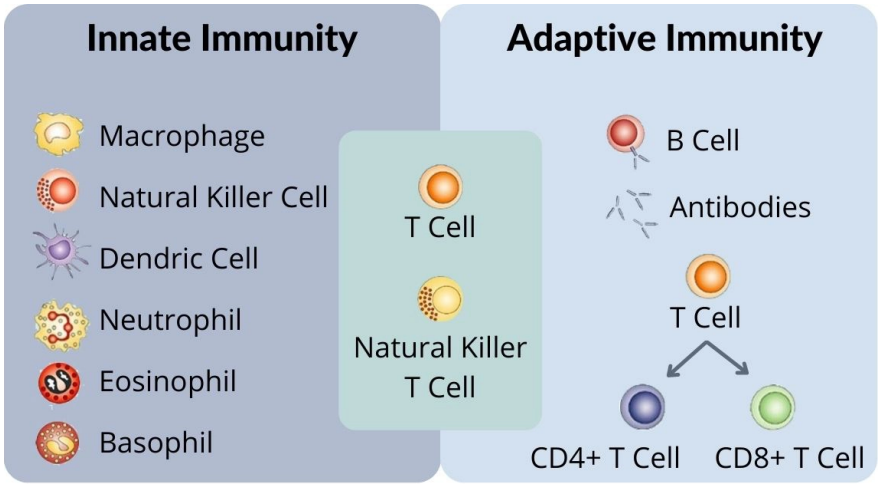


Figure 1.5 Innate vs. Adaptive Immunity.

which act as "killers" and communication agents. A complement system is also involved in this process, where a group of proteins flow freely into the bloodstream to recognise foreign entities before reacting to them. These proteins can also bind together to form a stronger cell-membrane in order to attack the intruder.

Adaptive Immune System: Similarly, the ADIS provides the host organism with defender cells by producing antibodies. Although this one is a more complex system, it is of special interest since this is where the AIS paradigm concept relies. The ADIS is composed by two subsystems: a Humoral response and a Cellular response. The white blood cells in the Humoral response are known as T-helper cells (or T-cells), and their objective is to bind with a particular antigen and then activate another type of white blood cells known as B-cells (or B lymphocytes). B-cells, in turn, produce antibodies that react towards the antigen trough a Negative Selection (NS) process. Lastly, the Cellular response focuses on binding the T-cells to antigens, and also releasing cytokines, which activate macrophages for intruder destruction. Both subsystems are shown in Figure 1.6 [39].

Negative Selection: An important characteristic of the ADIS is that it has the capability of detecting antigens while not reacting towards the self cells, leading to a discrimination-type process. This process is known as Negative Selection, although Positive Selection (PS) is also used for achieving the same purposes [40],[41]. However, PS is usually more computational

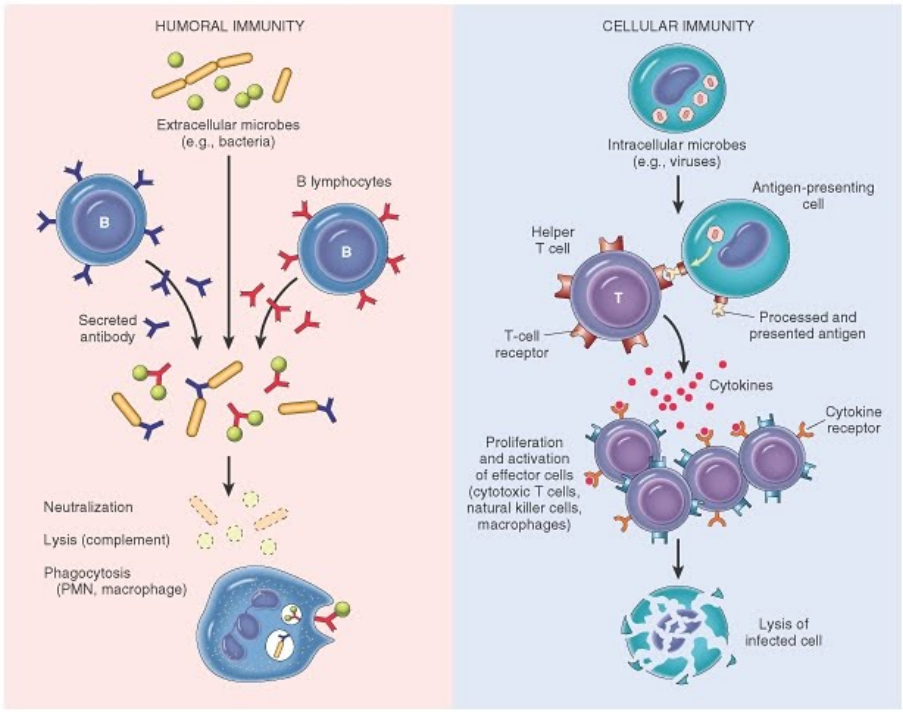


Figure 1.6 Humoral and Cellular Immunity.

expensive since the whole set of antibodies must be tested before declaring an external agent to be abnormal, therefore NS is the selected scheme by default [36]. Here, phagocytes (an immunological type of cells) are in charge of marking external intruders whose biochemical markers differ from the host's organism. In this way, the discrimination process can be carried out which will mark the difference between the self cells and non-self cells. For the purposes of this thesis, the terms "non-self", "detectors", and "antibodies" will be used interchangeably as will be explained further. The concept of NS is illustrated in Figure 1.7 [41], where all the process is biologically developed inside the thymus.

AIS for UAV Health Monitoring: Under the mentioned concepts, the BIS can be translated into the AIS paradigm by considering that the entire system (UAV in this case) can be represented by a set of features $\mathfrak{F} = \{\phi_i | i = 1, 2, ...N\}$ that capture the dynamics of the system at nominal operating conditions over a range of certain flight conditions [41]. These features will contain the nominal dynamic fingerprint of the UAV, and they will be used to build the self and non-self hyperspaces of the aircraft by a process of clustering

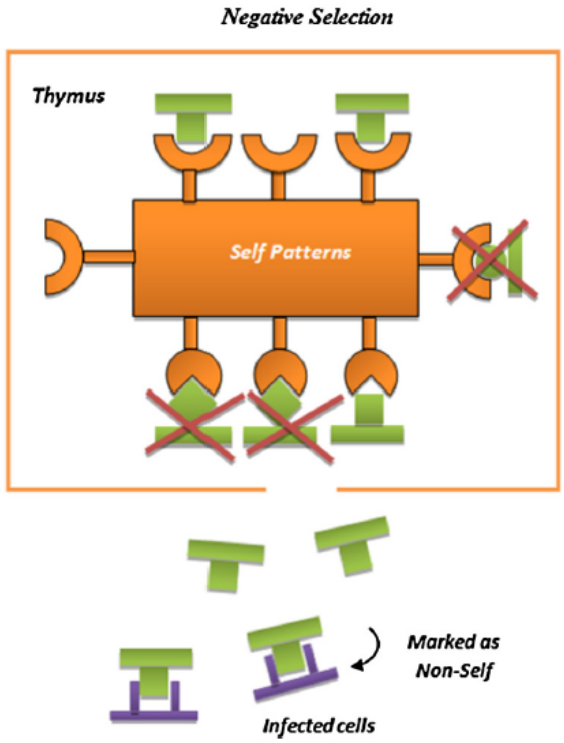


Figure 1.7 Negative Selection (NS) Concept.

and optimization of the acquired flight data. The hyperspace can be N-dimensional, but usually it is represented as a two or three dimensional space for visualization purposes and as such, these representations are known as projections. Furthermore, it is recommended that a maximum of 10 features must be selected so dimensional problems can be avoided. The set of feature can include dynamical sates of the aircraft or other type of variables related to it according to the analysis.

For clarification into the presented insight of the AIS, a study conducted by M.G. Perhinschi et al.[42] shows how the paradigm can be used to evaluate the flight envelope of the vehicle after some abnormal conditions (failures) are detected and identified for a NASA's Generalized Transport Model Aircraft. First, the authors define the flight envelope as the hyperspace of all achievable or desired variables, and they select a set of features that include roll acceleration error and roll rate commanded. Then, they use these variables to present an example of a 2D projection where they have built the self and non-self hyperspaces. After this, they have

conducted one flight at nominal conditions and then one flight with some type of failure, and then they have plotted the test data within the projections, as shown in Figure 1.8 [42]. Notice that the data has been normalized mainly for reducing computational effort.

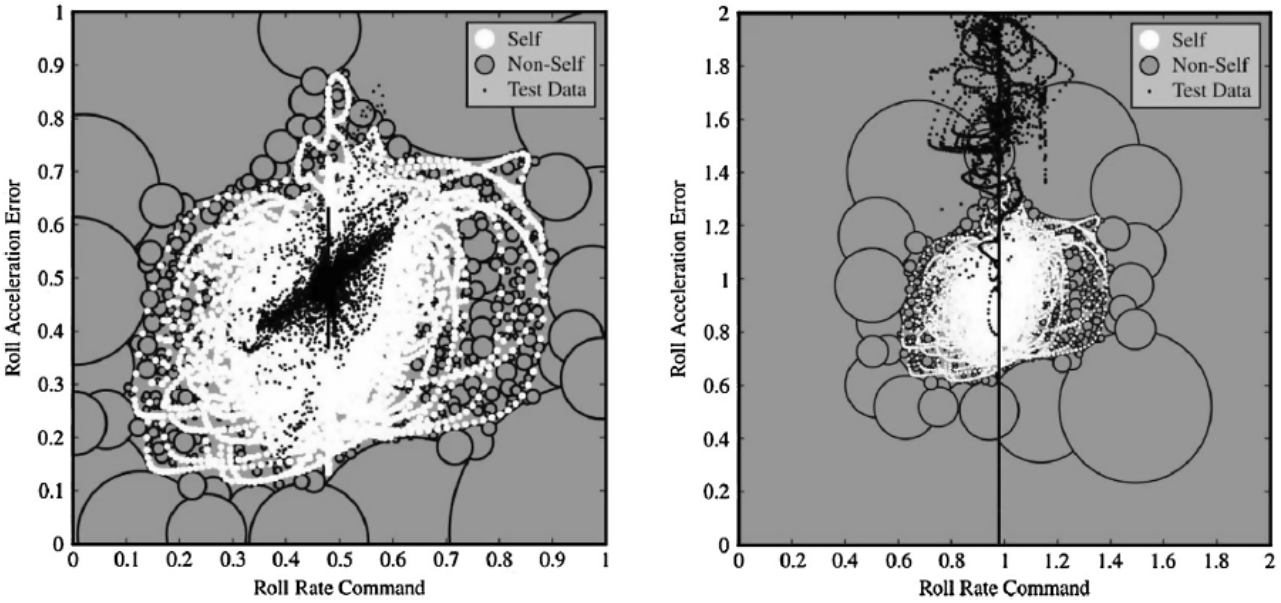


Figure 1.8 Nominal (left) vs. Abnormal (right) Dynamic Fingerprint.

For visualization purposes, the authors have used spheres to represent the hyperspaces of the self and the non-self, though this is not the only method available to be used. It is interesting to notice that when the aircraft is flying at nominal conditions, all the test data points fall within the self which means no abnormal flight conditions are present as there is lack of "intruders" within the non-self. In contrast, when the failure is introduced, the dynamic fingerprint of the aircraft stays unaltered, but the test data points fall within the non-self, meaning that there are intruders inside the antibodies area and these are being detected by the algorithm, as expected. For instance, the AIS paradigm has allowed the detection of an abnormal condition, and then an identification and evaluation process is carried out in order to characterize exactly which system or subsystem of the aircraft has failed.

The AIS paradigm is a tremendously helpful tool that is being constantly improved and adapted to new studies involving artificial intelligence and machine learning. Not only it offers advantages of computational efficiency, but it allows the system to be fully represented by its unique dynamical characteristics which further allows the detection, identification, and evaluation phases to describe a failure. More studies have implemented this technique such as the one done by Garcia[43], where a similar analysis is implemented into a quad-rotor UAV. The mathematical background and details of this theory is presented in chapter in 4.

1.2 Thesis Objectives

This thesis aims to present a novel mission planning architecture based on the combination of a Health Monitoring System (HMS) strategy that relies on the AIS framework for failure detection and identification, and an optimal path planning algorithm based on the optimized version of the Rapidly-Exploring Random Tree (RRT*) algorithm, such that safe trajectories can be generated considering that the fixed-wing UAV is experiencing a reduced flight envelope. The ultimate goal is to present an architecture that allows an increase in the autonomy of the vehicle under abnormal flight conditions, reducing the risk of potential collision and damage while flying in a modeled urban environment that uses 3D occupancy maps to simulate fixed obstacle avoidance. A simulation environment that uses Matlab/Simulink software is used along with a 6-DOF model of a fixed-wing called Rascal 110 commanded by a Nonlinear Dynamic Inversion Controller (NLDI) for tracking the safe generated trajectories.

1.3 Thesis Outline

First, theoretical background of occupancy maps is presented in Chapter 2 for modeling a 3D urban environment. Then, the RRT* path planner algorithm is presented in Chapter 3. The HMS is shown in Chapter 4 for failure detection and identification. Chapter 5 describes the simulation environment and the aircraft dynamics used for the simulation. Chapter 6 presents the results of successful replanning missions and a performance analysis of the generated safe trajectories. Finally, Chapter 7 lists conclusions and scope of future work.

2 Occupancy Maps

2.1 Theoretical Background

The simulation environment for this study is created using Matlab/Simulink software with the main objective of facilitating the interaction between the UAV dynamics and the environment around it. This will also be useful to pass the necessary information to the path planner for searching the free-state space where no obstacles are present and once the flight envelop is reduced due to an abnormal flight condition. The concept of 3D occupancy maps is used following the work presented by Wurm et al.[13] and Hornung et al. [14] as mentioned previously in subsection 1.1.1.

The main idea of occupancy maps is to provide an efficient-memory saving- 3D map of the flying area that can be easily uploaded to the on-board computer or microcontroller, and that contains enough details that can model static obstacles. The main advantage of building these type of maps is that they are based on a probabilistic 3D mapping framework which rely on structures called octrees, defined as hierarchical data structures for spatial subdivision in 3D which greatly saves computational memory and power[14].

A group of octrees leads to the development of an OctoMap mapping framework, in which a structure, similar to the branches of a tree, is subdivided into nodes such that each node represents the space contained in a cubic volume on a free three-dimensional space. In the literature, this volume is often called a voxel, and its size is used to determine the resolution of the map. The idea is illustrated in Figure 2.1 [14].

Furthermore, this voxel unit is subdivided into eight sub-volumes until a certain threshold is achieved which is usually the resolution that is desired. An important detail to consider is that the voxels not only consider the occupied space, but they also contain information about the free space and the unknown space which provides additional information to the UAV navigation and path planning algorithm. This allows the map to provide a more robust characterization of the position and space of the obstacles, therefore providing to the path planner bounds in which feasible trajectories can be created.

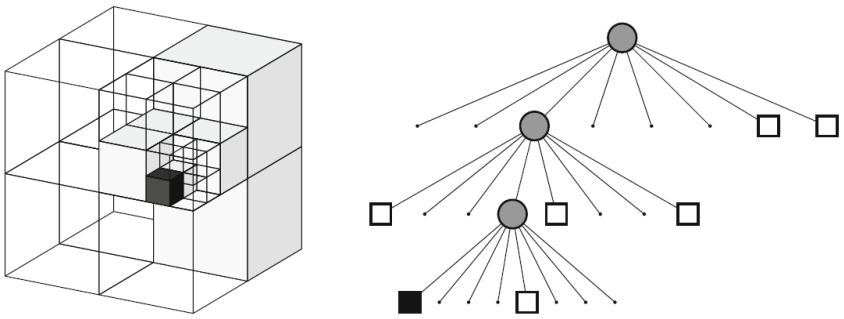


Figure 2.1 Octree Structure with Free (shaded white) and Occupied (black) cells represented as a Volume (left) and as Tree-Structure (right).

Based on this idea, a $700m \times 700m \times 700m$ 3D occupancy map is created with a resolution of $0.1m/cell$. The idea is to create a representation of an urban environment that can be commonly found in populated areas that contain buildings and streets of different sizes. The map is designed in a way that non-holonomic systems and holonomic systems are able to navigate the entire state-space, although only a fixed-wing UAV will be used for the purposes of this study. The generated map that will be used throughout this work is shown in Figure 2.2. Notice that the color of the voxels is strictly height-based, and it has nothing to do with the volumetric space occupied. For this case, the resultant memory occupied by the entire map is 8.232 MB which does not represent a memory size restriction to the OBC.

2.2 Obstacle Avoidance

Usually, 3D occupancy maps are based on probabilistic modeling of the map, meaning that known obstacles do not necessarily present a 100 % of probable collision with the vehicle. Especially when dynamic obstacles are being considered in representation of other terrestrial and airborne objects/animals typically present on these scenarios such as cars or birds, models of probabilistic functions will be needed to account for these effects. In fact, a dynamic environment will require a discrete occupancy probabilistic model to account for the changes in every time step of the simulation which will increase computational effort.

However, since only static obstacles will be considered to model the environment, a much simpler method for determining occupied volumes in the state-space is used. This method is

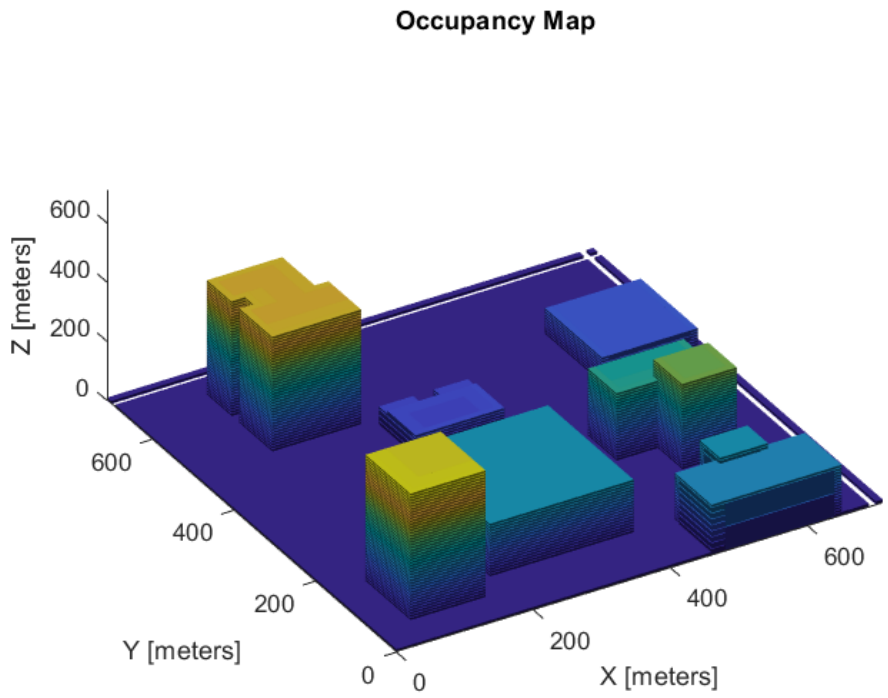


Figure 2.2 Generated 3D Occupancy Map for Urban Environment Simulations.

uses the Boolean property, where a 3D occupancy grid is created and assigned a true value (1) for the occupied spaces, a false value for free spaces (0), and a negative value for unknown spaces (-1). This allows to simplify greatly the construction of the map since the path planner will receive information about which volume is totally occupied, representing a 100% of probable collision, and a 0% of probable collision for the free spaces. Another advantage of using the Boolean property is that it allows similar voxels to be pruned from the state space, which results in a reduction of nodes needed in the octree structure. For implementation purposes, pruning methods greatly reduce the memory needed to store the map information, especially when the map is pre-uploaded in the vehicle instead of using some sensor such as LIDAR to create an on-line map of the obstacles. By these means, Figure 2.3 represents the complete obstacle avoidance map as seen by the vehicle. The red dots represent the unknown state-space of the map and their purpose is to provide a physical boundary limitation that would be needed by the path planner. Lastly, the red zones represent the flyable zone in which the numerical simulations are performed. This area includes all the state-space above

the defined obstacles.

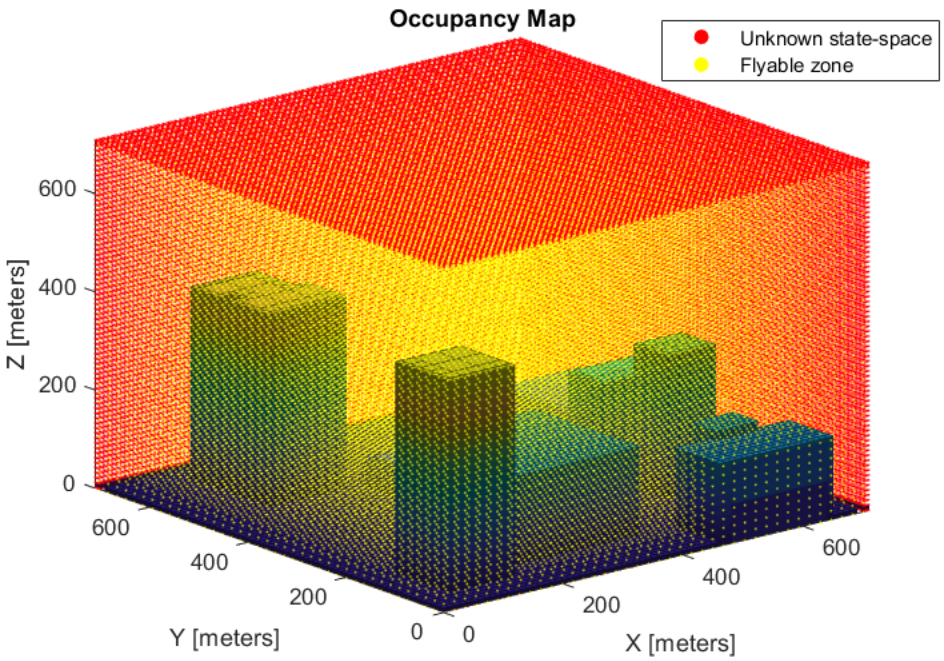


Figure 2.3 3D State-Space Definition for Obstacle Avoidance.

3 Trajectory Generation for Autonomous Navigation

This chapter presents the background theory of the selected RRT* path planner as mentioned in the Introduction. The most important concepts for kinodynamic planning will be presented as well as the necessary algorithms to plan and optimize the path based on some flight envelope considerations that will allow the generation of safe trajectories due to specific failures inserted in the system.

3.1 Requirements for Autonomous Navigation

Before jumping to the description of the path planner algorithms, it is necessary to formally define the properties that the planner must have for achieving the generation of safe paths during an on-line process. The most important properties were mentioned in Table 1.3, however, formal definitions of kinodynamic planning, probabilistic completeness, and asymptotic optimality are required in order to fully understand why the RRT* algorithm is the best choice for the present application.

3.1.1 Differential Constraints for State Propagation

Let $\chi = (0,1)^d$ define the state space or configuration space where $d \in \mathbb{N}$ is the dimension of the state space (i.e. $d = 3$ for this case). In addition, let the obstacle space χ_{obs} be defined as an open set such that $\chi \setminus \chi_{obs}$, and the obstacle-free space χ_{free} as the closure of the set $\chi_{free} = cl(\chi \setminus \chi_{obs})$. Then, a feasible path σ is found if it is a collision-free path with $\sigma(0) = x_{init}, and\, \sigma(1) \in cl(\chi_{goal})$, where x_{init} is the initial condition inside the obstacle-free space and χ_{goal} is the goal region.

With this in mind, recall that the fixed-wing UAV is a nonlinear system of differential equations (as further described in Chapter 5) that can generally be described by Equation 3.1, where $x \in \chi$ are the states of the system, and $u \in U$ are the set of allowable controls or inputs.

$$\dot{x} = f(x, u) \tag{3.1}$$

This equation imposes restrictions on the behaviour of the system which relies on its natural physical nature. For instance, the kynodynamic problem relies on being able to integrate the aforementioned equation over a time interval $[t, t + \Delta t)$ in order to obtain $x(t + \Delta t)$ after a certain control input is applied $[u(t')|t \leq t' \leq t + \Delta t]$. This can be achieved over continuous or discrete time intervals by using integration techniques such as the Runge-Kutta method. However, finding the appropriate feasible path depends entirely whether the integration can be performed or not, which implies that the non-holonomic constraints of the fixed-wing must be satisfied for obtaining a solution to the problem. This concept is of special importance, as it narrows the general path planning problem into a specific non-holonomic planning problem that allows the use of a more efficient trajectory generator for the fixed-wing.

3.1.2 Probabilistic Completeness

For assuring that the path planner arrives at a solution, it must be probabilistic complete. Especially if there are failures that reduce the flight capabilities, it is of extreme importance to try to guarantee that the flight can be still carried by generating another trajectory that connects the same waypoints. As mentioned before, probabilistic completeness is defined as the probability of finding a solution when the number of node samples goes to infinity [32].

Let $(\chi_{free}, x_{init}, \chi_{goal})$ define any robustly feasible path encountered by the planner. Then the limit in Equation 3.2 must exist and must be equal to 1.

$$lim_{n\to\infty}\mathbb{P}(\{\exists x_{goal} \in V_n \bigcap \chi_{goal} \; such \; that \; x_{init} \; is \; connected \; to \; x_{goal} \; in \; G_n\}) = 1 \qquad (3.2)$$

where: $G_n = (V, E)$ is the set of the entire vertex V_n (nodes) and edges E (straight lines connecting the nodes) returned by the planner.

If the planner finds more than one feasible solution, then this definition can be slightly modified as below.

$$lim_{n\to\infty}\mathbb{P}\left(\sum_n \sigma \neq \emptyset\right) = 1 \qquad (3.3)$$

where: $\sum_n \sigma$ are all the feasible obstacle-free paths encountered.

3.1.3 Asymptotic Optimality

Within the same line of thought, a path planner is considered to be asymptotic optimal if, for any feasible path encountered $(\chi_{free}, x_{init}, \chi_{goal})$, and cost function $c : \sum \rightarrow \mathbb{R}_{\geq 0}$, the limit in Equation 3.4 exists and is equal to the optimal (less) cost c^*, and the probability of converging into it is equal to 1.

$$\mathbb{P}\left(lim_{n\rightarrow\infty} sup\, min_{\sigma\in\sum\sigma}\left\{c(\sigma)\right\} = c^*\right) = 1 \tag{3.4}$$

Notice that the expression above implies implicitly that the algorithm must have first probability completeness before being asymptotic optimal. Moreover, it is emphasized that given a certain trajectory, the algorithm will absolutely converge or not converge at all into the optimal solution, meaning that the probability of Equation 3.4 is always zero or one, and there is no mid-point in the calculation.

Gammell and Strub [32] also introduce an additional backup definition of asymptotic optimality in probability. They state that as the number of samples goes to infinity, the probability of finding a solution being worse than the optimal solution goes to zero. This is in accordance with the first definition presented. Let ϵ represent any positive constant $\epsilon > 0$. Then, according to the authors, the following limit exists and the probability is equal to zero.

$$\forall\epsilon > 0,\ lim_{n\rightarrow\infty}\ sup\mathbb{P}\ \left[\left(min_{\sigma\in\sum\sigma}\{c(\sigma)\} - c^*\right) > \epsilon\right] = 0 \tag{3.5}$$

Equation 3.5 is an equivalent definition of Equation 3.4. Demonstrations of probability completeness and asymptotic optimality specifically for the Rapidly-Exploring Random Tree algorithms is an extensive procedure and out of the scope of this study. However, their basic definitions are presented here in order to have a formal insight of the nature of the following algorithms shown in the next section.

3.2 Rapidly-Exploring Random Trees (RRTs)

A question arises about if there is a possibility of generating a trajectory based on the restricted state propagation of the vehicle such that a feasible and save path is explored. The existence of this unique path will guarantee that the vehicle can still complete the mission, thereby reducing the risks of potential harm to civilians and damage to property.

Following this idea, the concept used by the Rapidly-Exploring Random Tree (RRT) path planning algorithm is used as a basis. First developed by La Valle [44] and further improved by Karaman and Frazzoli [31], this algorithm allows the exploration of the entire state space between two states x_1, x_2 in the three-dimensional space, therefore allowing the choice of a single path out from multiple possible paths that connect the initial position of the UAV and the final position desired. The selection of the best path, in this case, is based entirely on a cost function related to the Euclidean distance between the randomly generated nodes shown in Table 1.1. The Euclidean distance is defined to be $5\,m$ over all the results presented in this thesis. This algorithm is based on single-query applications, and its pseudo-code is presented in Figure 3.1 [44],[31].

```
1 V ← {x_init}; E ← ∅;
2 for i = 1, ..., n do
3     x_rand ← SampleFree_i;
4     x_nearest ← Nearest(G = (V, E), x_rand);
5     x_new ← Steer(x_nearest, x_rand) ;
6     if ObtacleFree(x_nearest, x_new) then
7         V ← V ∪ {x_new}; E ← E ∪ {(x_nearest, x_new)} ;
8 return G = (V, E);
```

Figure 3.1 Original RRT Algorithm with Collision Checking.

When initialized, the initial state previously defined as x_{init} contains a single initial vertex with no edges. Then, at each iteration, new randomly nodes are generated in the free space, χ_{free}. Once a nearest node $x_{nearest}$ is found based on the Euclidean distance to the parent node, an attempt to connect both nodes is made. The connection is successful if, and only if, there is an obstacle-free edge E between $x_{nearest}$ and x_{random}. Assuming the connection is

successful, then the nodes are added to the vertex set $x_{random} \in V$ with their own edge set E. The process is repeated until the initial pose and final pose are connected after n iterations. Finally, the algorithm returns the desired tree as a graph represented by the entire set of vertex and edges $G = (V, E)$.

As an example, consider the selected poses in Table 3.1 with their respective altitudes and desired headings, which are drawn on the urban 3D map (defined in Chapter 2) in Figure 3.2. Here, the red node is the start position, and pose #3 is the final desired goal. Taking into account that nominal conditions of roll angle $\phi_{nominal} = \pm 50\,\text{deg}$, and flight path angle $\gamma_{nominal} = \pm 45\,\text{deg}$ with a maximum of 2000 iterations is desired, the tree is generated with 275 nodes, $n = 476$ iterations at an elapsed time of $t = 9.16s$.

Table 3.1 Predefined coordinates for the example of the original RRT path generation.

	Cartesian Coordinates			
Pose	**X (m)**	**Y(m)**	**Z (m)**	**Heading (deg)**
Start	600	620	60	20
#1	470	430	80	-180
#2	170	450	60	-180
#3	100	200	50	0

Figure 3.3 shows the result of the path planning, where the tree's node and edges (highlighted in yellow) attempt to connect the desired initial position with all the poses to form a trajectory (highlighted in green). Notice that, in general, the path is not suitable to be followed by a fixed-wing or any non-holonomic system due to its sharp turns and clear formation of "cycles". For instance, improvements need to be added considering that not even the most cost-efficient path is selected.

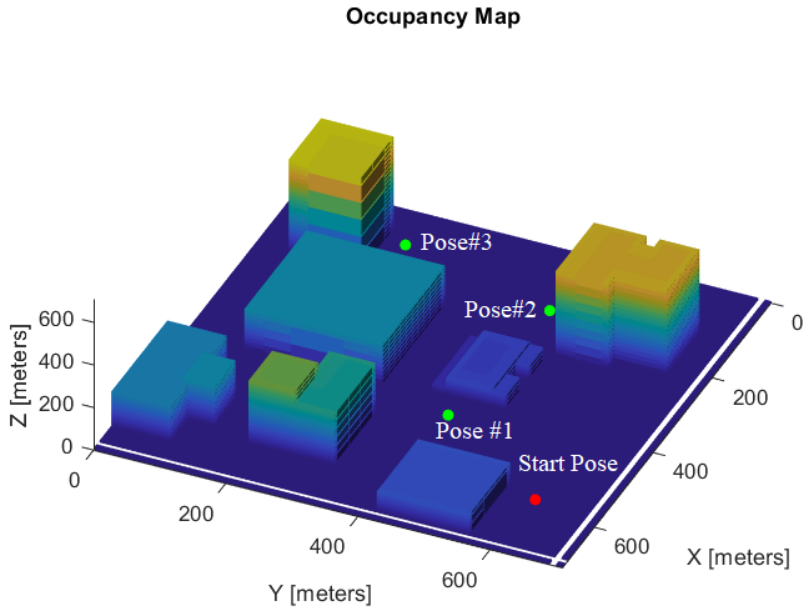


Figure 3.2 Selected Poses for the Example of the Original RRT Path Generation.

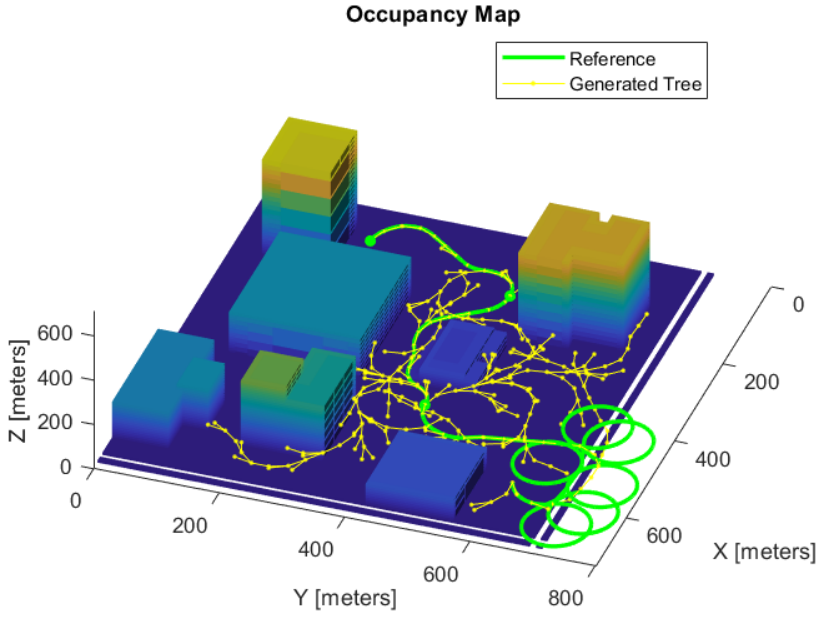


Figure 3.3 Example of Original RRT Path Generation.

3.2.1 Rapidly-Exploring Random Graph (RRG)

The RRG path planner introduces a modification into the original RRT algorithm that has to do with an improvement of the wiring of the nodes inside the tree. Figure 3.4 [31],[44] shows the pseudo code of the RRG. Similarly as the RRT, the algorithm starts searching the state space and adding new nodes into the vertex set if the connection is successful, and collision checking is also performed at every iteration. However, in line 7, a function is introduced that forces a new way of connecting the nodes: every time a new point x_{new} is added to the vertex set in the previous step, new connections are attempted from all the other nodes in V that are within a certain ball radius.

For defining this ball radius, let $card(V) \leq n+1$ and $\gamma_{RRG} = 2(1+\frac{1}{d})^{\frac{1}{d}} \left(\frac{\mu(\chi_{free})}{\zeta_d}\right)^{\frac{1}{d}}$, where $\mu(\chi_{free})$ denotes the volume of the obstacle-free space, and ζ_d denotes the volume of the unit ball in the $d = 3$ Euclidean dimensional space. Then, the radius is defined as a minimum value in Equation 3.6.

$$r(card(V)) = min\left\{\left(\frac{\gamma_{RRG}(log(card(V))}{card(V)}\right)^{\frac{1}{d}}, \eta\right\} \tag{3.6}$$

where: the constant η is defined as $\eta \geq \|z - x\|$ which comes from the steering function of the vehicle (though it is a parameter that will not be tuned in this study).

1 $V \leftarrow \{x_{\text{init}}\}$; $E \leftarrow \emptyset$;
2 **for** $i = 1, \ldots, n$ **do**
3 $\quad x_{\text{rand}} \leftarrow \texttt{SampleFree}_i$;
4 $\quad x_{\text{nearest}} \leftarrow \texttt{Nearest}(G = (V, E), x_{\text{rand}})$;
5 $\quad x_{\text{new}} \leftarrow \texttt{Steer}(x_{\text{nearest}}, x_{\text{rand}})$;
6 $\quad$ **if** $\texttt{ObtacleFree}(x_{\text{nearest}}, x_{\text{new}})$ **then**
7 $\quad\quad X_{\text{near}} \leftarrow \texttt{Near}(G = (V, E), x_{\text{new}}, \min\{\gamma_{\text{RRG}}(\log(\text{card}\,(V))/\,\text{card}\,(V))^{1/d}, \eta\})$;
8 $\quad\quad V \leftarrow V \cup \{x_{\text{new}}\}$; $E \leftarrow E \cup \{(x_{\text{nearest}}, x_{\text{new}}), (x_{\text{new}}, x_{\text{nearest}})\}$;
9 $\quad\quad$ **foreach** $x_{\text{near}} \in X_{\text{near}}$ **do**
10 $\quad\quad\quad$ **if** $\texttt{CollisionFree}(x_{\text{near}}, x_{\text{new}})$ **then** $E \leftarrow E \cup \{(x_{\text{near}}, x_{\text{new}}), (x_{\text{new}}, x_{\text{near}})\}$
11 **return** $G = (V, E)$;

Figure 3.4 RRG Algorithm with Collision Checking.

Although this modification does not solve the problem of the formation of cycles, it is necessary in order to understand the optimization introduced by the RRT* algorithm.

3.2.2 Rapidly-Exploring Random Tree* (RRT*)

Formally speaking, the RRT* algorithm is an extended version of the RRG algorithm which, at the same time, is an extension of the RRT algorithm. Again, the RRT* algorithm searches the nodes and adds them into the vertex set in the same way as the RRT and RRG, and it also uses the ball radius to make new connections. However, the novelty of RRT* is the introduction of two cost-optimization parameters that limit the feasible connections that the planner can make in each iteration:

- An edge connecting X_{near} to x_{new} will be added to the edge set if, and only if, it follows a minimum cost path.
- If several edges are connected from x_{new} to X_{near}, then the algorithm calculates the cost of all the connected edges, and then picks up the one with less cost and maintains it while it deletes the rest of the edges.

By doing these modifications, a rewiring of the tree is made such that the formation of cycles is avoided, and the amount of required processing power is reduced as cost-efficient trajectories are generated. This allows the online implementation of the planner to be feasible in a real-world application. Figure 3.5 [31] shows the entire RRT* algorithm, which is the main path planner used in this thesis to generate feasible safe trajectories under a restricted flight envelope of the fixed-wing Rascal 110. Notice that until line 8, the algorithm is similar to the RRG. Lines 9, 11-12, and 15-16 introduce the cost functions for the optimization. These functions are defined as follows: let $Line(x_1, x_2)$ be the straight line connection between points $x_1, x_2 \in \mathbb{R}^3$, and $c(Line(x_1, x_2)$ the cost of the connection path; then, the algorithm will connect the nodes along minimum-cost paths and then rewire the tree by using cost-additive functions of the form $c_{min} = Cost(x_{1,2}) + c(Line(x_{1,2}, x_{1,2}))$. Figure 3.6 clearly shows the improvement of the path in contrast with Figure 3.3, and the cost-optimum path is shown.

$V \leftarrow \{x_{\text{init}}\}$; $E \leftarrow \emptyset$;
for $i = 1, \ldots, n$ **do**
  $x_{\text{rand}} \leftarrow \texttt{SampleFree}_i$;
  $x_{\text{nearest}} \leftarrow \texttt{Nearest}(G = (V, E), x_{\text{rand}})$;
  $x_{\text{new}} \leftarrow \texttt{Steer}(x_{\text{nearest}}, x_{\text{rand}})$;
  if $\texttt{ObtacleFree}(x_{\text{nearest}}, x_{\text{new}})$ **then**
    $X_{\text{near}} \leftarrow \texttt{Near}(G = (V, E), x_{\text{new}}, \min\{\gamma_{\text{RRT}^*}(\log(\text{card}\,(V))/\,\text{card}\,(V))^{1/d}, \eta\})$;
    $V \leftarrow V \cup \{x_{\text{new}}\}$;
    $x_{\min} \leftarrow x_{\text{nearest}}$; $c_{\min} \leftarrow \texttt{Cost}(x_{\text{nearest}}) + c(\texttt{Line}(x_{\text{nearest}}, x_{\text{new}}))$;
    foreach $x_{\text{near}} \in X_{\text{near}}$ **do** // Connect along a minimum-cost path
      if $\texttt{CollisionFree}(x_{\text{near}}, x_{\text{new}}) \wedge \texttt{Cost}(x_{\text{near}}) + c(\texttt{Line}(x_{\text{near}}, x_{\text{new}})) < c_{\min}$ **then**
        $x_{\min} \leftarrow x_{\text{near}}$; $c_{\min} \leftarrow \texttt{Cost}(x_{\text{near}}) + c(\texttt{Line}(x_{\text{near}}, x_{\text{new}}))$
    $E \leftarrow E \cup \{(x_{\min}, x_{\text{new}})\}$;
    foreach $x_{\text{near}} \in X_{\text{near}}$ **do** // Rewire the tree
      if $\texttt{CollisionFree}(x_{\text{new}}, x_{\text{near}}) \wedge \texttt{Cost}(x_{\text{new}}) + c(\texttt{Line}(x_{\text{new}}, x_{\text{near}})) < \texttt{Cost}(x_{\text{near}})$ **then** $x_{\text{parent}} \leftarrow \texttt{Parent}(x_{\text{near}})$;
      $E \leftarrow (E \setminus \{(x_{\text{parent}}, x_{\text{near}})\}) \cup \{(x_{\text{new}}, x_{\text{near}})\}$
return $G = (V, E)$;

Figure 3.5 RRT* Algorithm with Collision Checking.

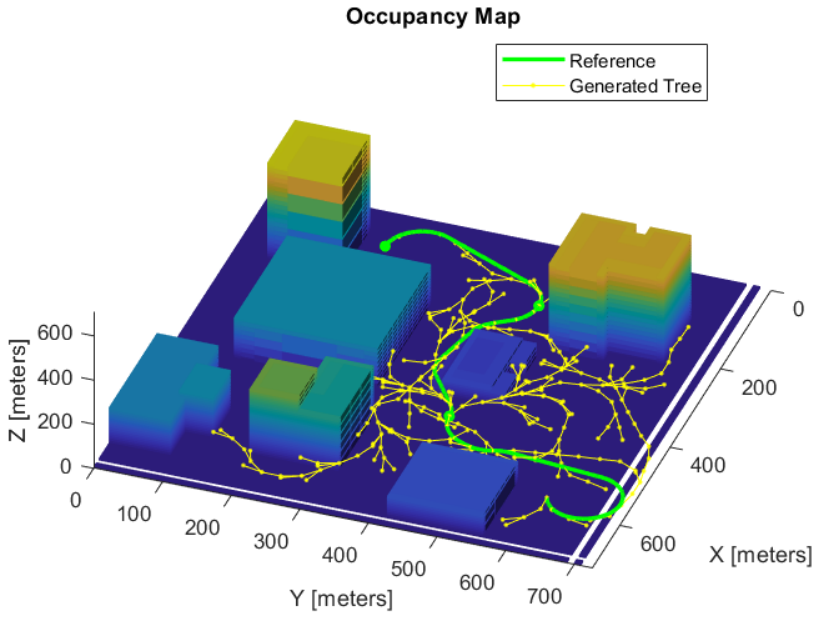


Figure 3.6 Example of RRT* Path Generation based on Table 3.1.

3.3 Dubins Paths for Smoothing

In the previous examples, notice that the green trajectory generated has always been smooth in the sense that the yellow branches of the generated tree have not been directly used to highlight the valid trajectory. This is because the RRT and RRT* algorithms have been merged with what is known as the Dubins Airplane Path generation, firstly proposed in 1957 by L.E. Dubins [33]. If the path planner would not consider the kynodynamic problem described in subsection 3.1.1, then there would not be a need of merging the Dubins Paths with the planner. However, since the purpose is to provide the best trajectory possible in terms of non-holonomic constraints, then the Dubins Airplane Paths concept is applied into each segment of the generated tree with the main goal of transforming sharp-edge connections and turns into smooth arcs, as previously shown on Figures 3.3 and 3.6.

Within this line of thought, this section briefly describes the equations that the Dubins Path uses to smooth the trajectory. First, an insight to the Dubins Car Paths is explained as this is needed for understanding the Dubins Airplanes Paths. Lastly, decision making capabilities of this algorithm is explained within the 2D and 3D dimensions.

3.3.1 Dubins Car Paths

The Dubins car model has been widely use in robotics path planning due to its relatively simple methodology. Beard and McLain [7] have adapted the concept using modern notation, and they propose a new set of EOMs that describe the Dubins car model given by the set of Equations 3.7.

$$\begin{aligned} \dot{r}_n &= V cos(\psi) \\ \dot{r}_e &= V sin(\psi) \\ \dot{\psi} &= u \end{aligned} \tag{3.7}$$

where: $\dot{r}_n$ and $\dot{r}_e$ are the North and East velocities, and $\dot{\psi}$ is the rate of change of the heading. In this case, the minimum turning radius of the car is given by $R_{min} = \frac{V}{\bar{u}}$, where $|u| \leq \bar{u}$.

The algorithm consists of switching between orbit following and straight-line following. The basic concept of this method is shown in Figure 3.7 [6]. Here, the initial configuration defined by $\boldsymbol{z_s} = (z_{ns}, z_{es}, \psi_s)$ is connected by a straight line and an arc to a final configuration defined by $\boldsymbol{z_e} = (z_{ne}, z_{ee}, \psi_e)$ in four different possible ways. The arcs are formed once two tangent circles are located between two defined centers $\boldsymbol{c_l}$ or $\boldsymbol{c_r}$ around the pose. Notice that since there are four possible ways of connections between these configurations, the Dubins path must introduce also the Euclidean minimum cost function in order to select the best path between the two poses. In this case, the "Right-Straight-Right"(RSR) scenario would be the selected path. On the other hand, the "Left-Straight-Left"(LSL) scenario would be the least feasible path by simple inspection.

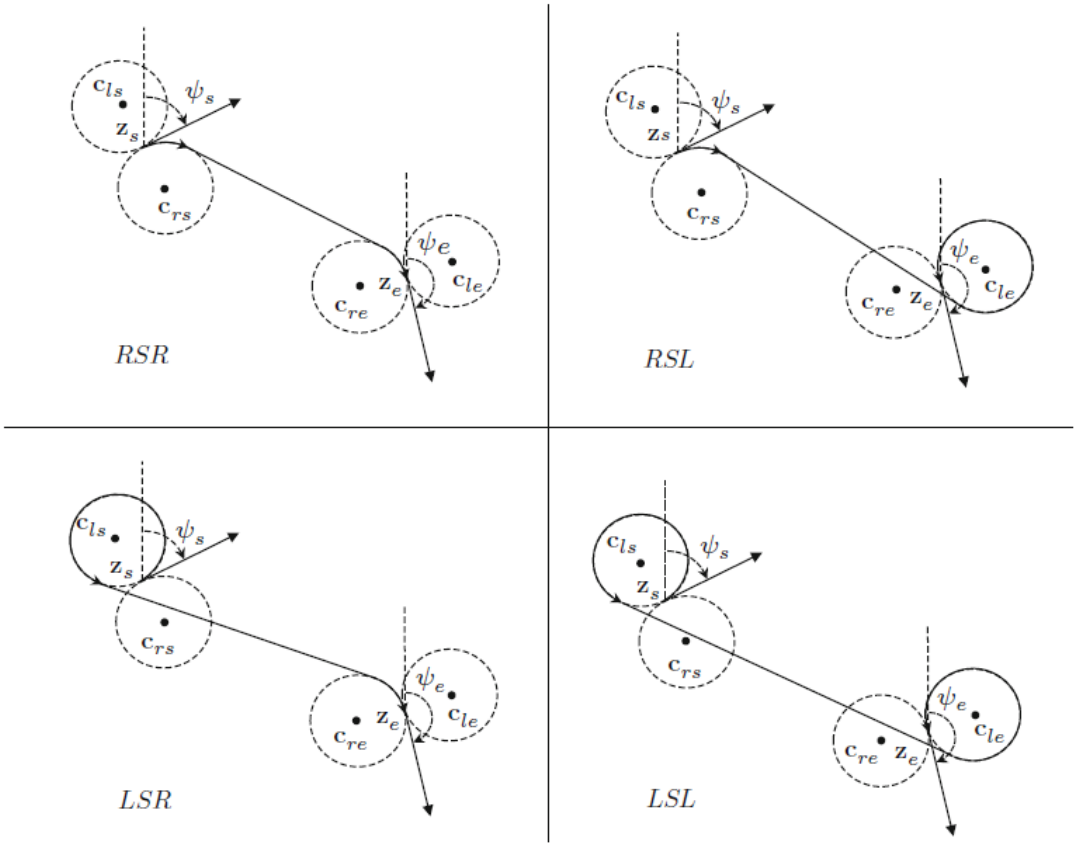


Figure 3.7 Four possible Paths connecting an Initial Pose $\boldsymbol{z_s} = (z_{ns}, z_{es}, \psi_s)$ and Final Pose $\boldsymbol{z_e} = (z_{ne}, z_{ee}, \psi_e)$ given a minimum turning Radius R_{min}.

3.3.2 Dubins Airplane Paths

Taking the concept of the Dubins Car Paths, then a third dimension can be added, and by redefining the minimum turning radius with modifications to the EOMs, the same concept can be applied to a fixed-wing UAV. Chitsaz and Lavalle [45] provide an extensive explanation of how the addition of the third dimension is added, and just the most important elements are presented here.

Consider Figure 3.8 [6] where the NED reference frame is used for deriving the kinematic model of a general fixed-wing UAV. For simplicity, wing effects (i.e. side-slip angle $\beta = 0$), aerodynamics, and engine thrust limits are neglected. Furthermore, assume the aircraft performs a coordinate turn condition as described by Equation 3.8.

$$\dot{\psi} = \frac{g}{V} tan(\phi) \tag{3.8}$$

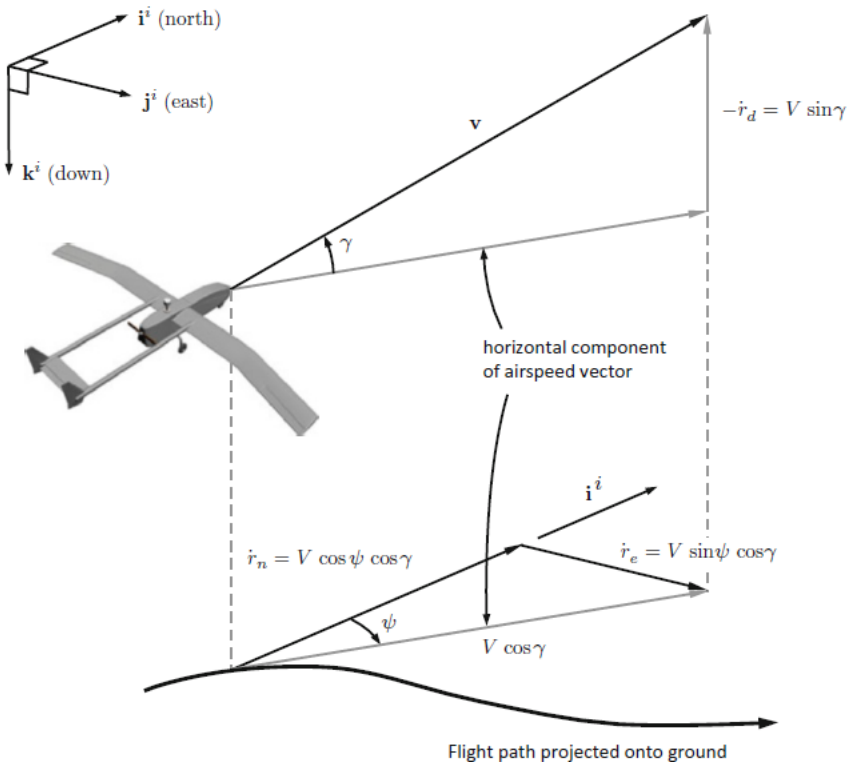


Figure 3.8 Basic Kinematic model of a generalized fixed-wing UAV.

Then, the kinematic model is

$$\begin{aligned} \dot{r}_n &= V cos(\psi) cos(\gamma^c) \\ \dot{r}_e &= V sin(\psi) cos(\gamma^c) \\ \dot{r}_d &= -V sin(\gamma^c) \\ \dot{\psi} &= \frac{g}{V} tan(\phi^c) \end{aligned} \tag{3.9}$$

where: γ^c and ϕ^c are the the commanded flight path angle and roll angle, respectively.

The advantage of this model is that the commanded bank and flight path angle depend entirely on the physical capabilities of the aircraft. As such, these limits are represented by the constraints on Equation 3.10 which pertain specifically for the Rascal 110 UAV at nominal flight conditions.

$$\begin{aligned} |\phi^c| &\leq \pm 50 deg \\ |\gamma^c| &\leq \pm 45 deg \end{aligned} \tag{3.10}$$

Moreover, the commanded angles can be sent by whichever controller or autopilot is used. For this case, a NLDI controller (described in Chapter 5 section 5.3.2) is in charge of performing this task. According to the authors, this model is suitable for high-level path-planning and path-following control design (i.e. the virtual controller that is implemented and explained on Chapter 5 section 5.3.1).

3.3.3 Decision making in 2D & 3D

One question remains in terms of which path is the best option to choose besides its cost. This is because the heading of the aircraft, or even an obstacle can prevent the planner to generate a feasible trajectory into a specific waypoint whether the UAV is flying at nominal or under failure conditions. It would not be logical for the planner to choose the path with minimum cost if this one is not feasible to be followed. Therefore, decision making capabilities are needed in 2D and 3D for guaranteeing the generation of a feasible safe trajectory in the

presence of obstacles and taking into account the initial and desired headings of the aircraft within each pose.

3.3.3.1 The 2D Problem

First, consider a trajectory in 2D similarly to the Dubins Car Paths, as shown in Figure 3.9 [6],[7]. Here, the vehicle is initially commanded to follow the orbit with center $\boldsymbol{c_s}$ at the start configuration. The orbit can have two directions: clockwise $\boldsymbol{\lambda_{s,e}} = 1$ or counter-clockwise $\boldsymbol{\lambda_{s,e}} = -1$. Based on the desired heading $\psi_{s,e}$ the planner chooses the direction of the orbit, and then the vehicle follows the orbit until it encounters the half-plane defined as $\mathcal{H}(\boldsymbol{w}_s, \boldsymbol{q}_s)$, where $\boldsymbol{w}_s$ is just a position on the half-plane and $\boldsymbol{q}_s$ is a unit vector orthogonal to the half-plane. Furthermore, the vehicle follows the straight path defined by $(\boldsymbol{w}_s, \boldsymbol{q}_s)$ until it reaches the other half-plane $\mathcal{H}(\boldsymbol{w}_l, \boldsymbol{q}_l)$ which is located opposite to the end half-plane of the end position $\mathcal{H}(\boldsymbol{w}_e, \boldsymbol{q}_e)$. Once the vehicle touches $\mathcal{H}(\boldsymbol{w}_l, \boldsymbol{q}_l)$, it is commanded to enter the orbit with center $\boldsymbol{c}_e$. Again, the direction of the orbit depends on the heading and finally, the algorithm stops when the $\mathcal{H}(\boldsymbol{w}_e, \boldsymbol{q}_e)$ plane is crossed.

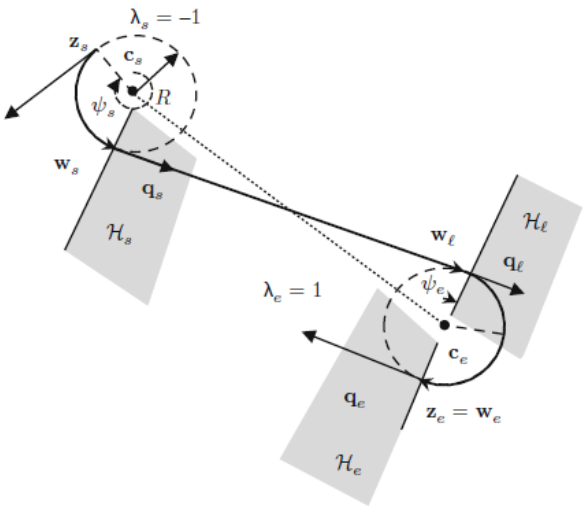


Figure 3.9 2D Dubins Path Decision-Making Capabilities.

It is important to highlight that not only 2D decision-making capabilities are provided using this algorithm, but the desired smoothness of the path is achieved due to the formed arcs.

3.3.3.2 The 3D Problem

For adding the altitude dimension, more parameters must be defined due to climb limits. Let $L_{car}(R_{min})$ represent the length of the Dubins Car Path which depends on the minimum turning radius R_{min} defined in Equation 3.11 for the Dubins Airplane.

$$R_{min} = \frac{V^2}{g} tan(\bar{\phi}) \tag{3.11}$$

where: $\bar{\phi}$ is the nominal or restricted roll angle limit.

Then, Owens et. al [46] classify the generation of the 3D trajectories in three categories, which strictly depend on the magnitude of the desired altitude and achievable flight path angle: *low-altitude*, *medium-altitude*, and *high- altitude*. For the purpose of this study, helices will be used to generate a feasible path for the UAV to reach certain altitudes, and the decision-making conditions are defined for each category as described below.

Low-Altitude 3D Dubins Path

A Dubins path is low-altitude if the condition of Equation 3.12 is satisfied.

$$|z_{de} - z_{ds}| \leq L_{car}(R_{min}) tan(\bar{\gamma}) \tag{3.12}$$

where: z_{ds} and z_{de} represent the down-component of the position in the NED frame of the start pose and final pose, respectively, and $\bar{\gamma}$ represents the limit of the flight path angle. Notice that the right term of this equation is the maximum altitude gain that can be achieved given a certain flight path angle $\pm\bar{\gamma}$. Then, the optimal flight path angle γ^* can be defined as

$$\gamma^* = tan^{-1}\left(\frac{|z_{de} - z_{ds}|}{L_{car}(R_{min})}\right) \tag{3.13}$$

while the length of the Dubins Airplane can be computed by

$$L_{airplane}(R_{min}, \gamma^*) = \frac{L_{car}(R_{min})}{cos(\gamma^*)} \tag{3.14}$$

Following the previous restrictions with the computation of Equations 3.13 and 3.14, feasible low altitude trajectories can be generated. An example is shown in Figure 3.10 [6],[7] for a "Right-Straight-Left"(RSL) scenario. It is important to mention that definitions for the initial and final headings $\psi_{s,e}$ are required such that the helices can be generated based on the most convenient orbit direction.

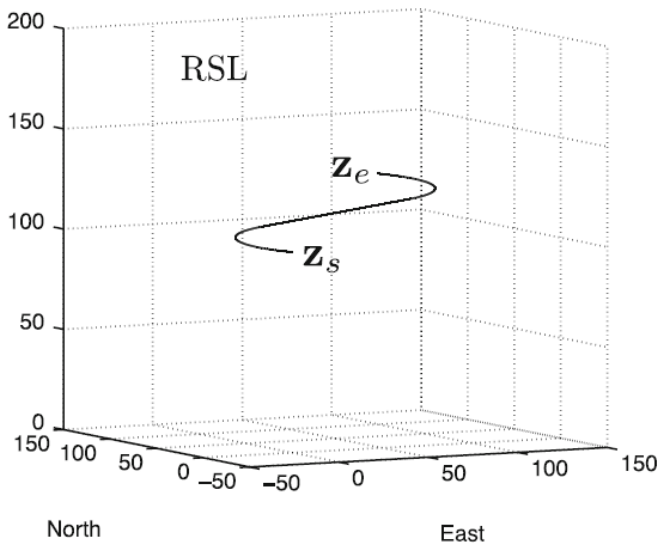


Figure 3.10 Dubins Airplane Path for Low-Altitude case.

Medium-Altitude 3D Dubins Path

Similarly as before, the restriction of Equation 3.15 must be true to categorize the path as a medium-altitude trajectory.

$$L_{car}(R_{min})tan(\bar{\gamma}) < |z_{de} - z_{ds}| \leq [L_{car}(R_{min}) + 2\pi R_{min}]tan(\bar{\gamma}) \tag{3.15}$$

Once this condition is true, an additional arc is inserted by reconsidering the 2D problem in Figure 3.9 [6],[7]. The reason for inserting this new arc is to allow for the UAV to arrive at the desired altitude by not extending too much the length of the path and considering the maximum climb limit of the vehicle in terms of its flight path angle defined by its nominal or restricted flight envelope, as will be discussed later.

Figure 3.11 shows the new trajectory with the new arc with center $\boldsymbol{c_i}$ that can be parameterized by an angle φ following Equation 3.16.

$$\boldsymbol{z}_i = \boldsymbol{c}_s + R(\varphi)(\boldsymbol{z}_s - \boldsymbol{c}_s) \tag{3.16}$$

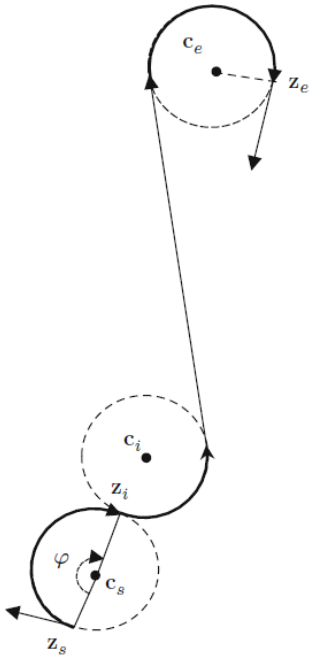


Figure 3.11 Dubins Airplane Path Parameters for Medium-Altitude case.

Using a bisection algorithm, the optimal parameter φ^* can be find by setting the condition of Equation 3.17.

$$L(\varphi^*)tan(\bar{\gamma}) = |z_{de} - z_{ds}| \tag{3.17}$$

This allows the computation of the length of the Dubins Airplane by Equation 3.18, similarly as for the low altitude case.

$$L_{airplane} = \frac{L(\varphi^*)}{cos(\bar{\gamma})} \tag{3.18}$$

Figure 3.12 [6],[7] shows an example of the medium altitude case where the end pose is at a higher altitude than the initial pose. Note that the introduction of the new arc imposes a new turning capability, and now the path can be defined as "Right-Left-Straight-Right"(RLSR) for this example. A great advantage of this method is that the smoothness of the path is hold despite the change of altitude and taking into account the climbing limits of the vehicle.

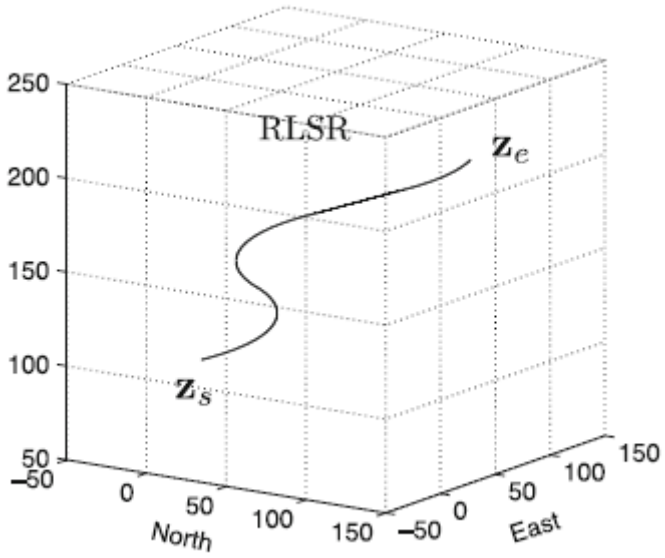


Figure 3.12 Dubins Path for medium altitude case.

High-Altitude 3D Dubins Path

The third and last case obeys the restriction shown in Equation 3.19.

$$|z_{de} - z_{ds}| > [L_{car}(R_{min}) + 2\pi R_{min}]tan(\bar{\gamma}) \tag{3.19}$$

Here, it is assumed that the desired climbing is not achievable within the normal flight path angle limits. Due to this, more helices will be used to achieve the desired altitude. Specifically, if the initial pose has a lower altitude that the final pose, the path will be extended such that climbing helices at the beginning of the trajectory will be generated. On the other hand, if the final pose has a lower altitude than the initial pose, descending helices

will be generated at the end of the trajectory. An example of this method is shown in Figure 3.13 [6],[7], where the starting pose has a lower altitude than the initial pose.

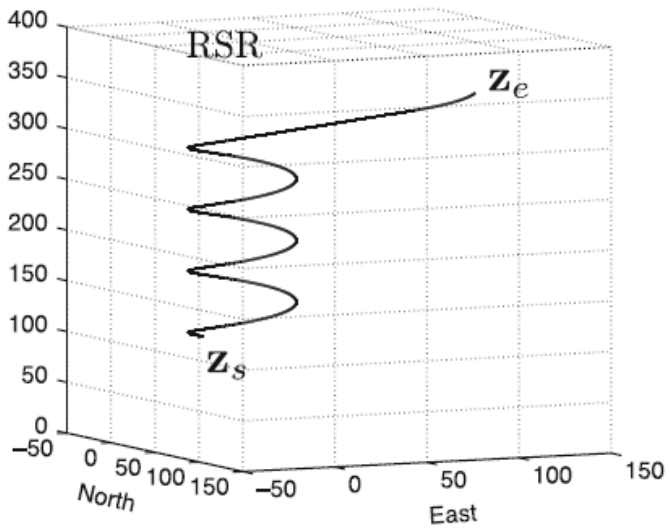


Figure 3.13 Dubins Path for high-altitude case.

For these type of cases, the number of turns of each helix k needed to completely achieve the desired altitude is defined by the following equation.

$$k = \left[\frac{1}{2\pi R_{min}} \left(\frac{|z_{de} - z_{ds}|}{tan(\bar{\gamma})} - L_{car}(R_{min}) \right) \right] \tag{3.20}$$

Once again, an additional constraint is defined such that the best turning radius R^* can be found. This can be achieved using, for example, a bisection algorithm that solves the following condition.

$$(L_{car}(R^*) + 2\pi k R^*) tan(\bar{\gamma}) = |z_{de} - z_{ds}| \tag{3.21}$$

Lastly, the high-altitude path length can be computed by Equation 3.22.

$$L_{airplane}(R^*, \bar{\gamma}) = \frac{L_{car}(R^*)}{cos(\bar{\gamma})} \tag{3.22}$$

Now, the 3D trajectory is feasible to be followed by the non-holonomic system after the algorithm classifies the additional third dimension of the poses in one of the categories described above. Consequently, a path can be generated by iterating multiple times the RRT* algorithm so multiple segments of trajectory can allow the fixed-wing to visit multiple waypoints or poses, and the Dubins path smoothness algorithm can then be applied to each segment of the trajectory to obtain a final safe trajectory. Figure 3.14 [46] shows the algorithm flow that is used to implement the high-altitude Dubins Airplane Paths. This logic is based on the half-planes previously defined in Figure 3.9.

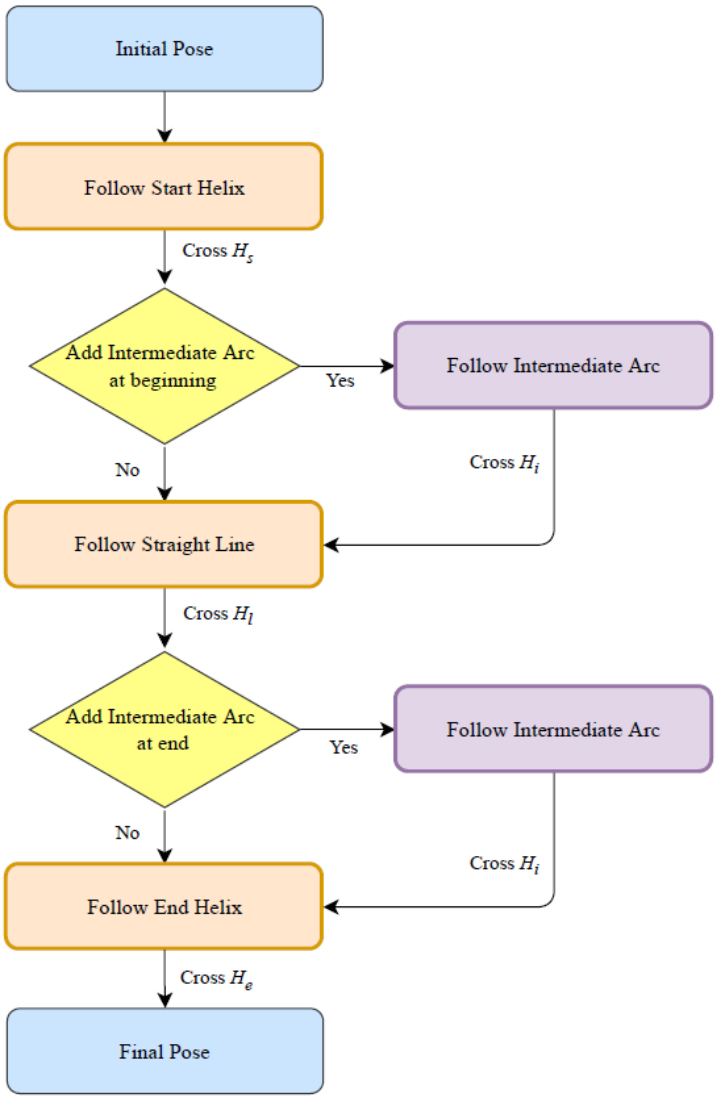


Figure 3.14 High-Altitude Decision-making logic.

4 Artificial Immune System Paradigm

This chapter briefly describes the use of the AIS paradigm for failure identification, detection, and evaluation of certain type of failures in the context of the safe trajectory generation for the fixed-wing Rascal 110. First, mathematical definitions of the features and the flight envelope at post-failure conditions are given as previously mentioned in the Introduction 1. Then, equations for estimating the flight envelope reduction in terms of restrictions in the roll angle is presented. Finally, a description of how the antibodies are generated within the AIS along with the failure detection, identification, and evaluation scheme is shown.

4.1 Post-Failure Flight Envelope Prediction

Let the set of *feature variables* $\mathfrak{F}$ a set of variables ϕ_i that completely capture and define the dynamical characteristics of the system (in this case the UAV):

$$\mathfrak{F} = \{\phi_i | i = 1, 2, ...N\} \tag{4.1}$$

Then, a subset of variables $\mathfrak{E}$ called *envelope relevant variables* (ERV) v_E can be defined which describes the flight envelope as the hyperspace of all the desired or achievable values [41], expressed as:

$$\mathfrak{E} = \{v_{Ei} | i = 1, 2, ...N_E\}, \mathfrak{E} \subset \mathfrak{F} \tag{4.2}$$

Notice that, in general, the selection of the feature variables will depend on which Abnormal Condition (AC) is being analyzed as will be further described. Furthermore, assume that each variable in the flight envelope has restricted nominal limits $[v_{Ei_{min}}, v_{Ei_{max}}]$. Then, intuitively, the set of ranges of all ERVs at post-failure conditions can be described as $[v_{Ei_{minF}}, v_{Ei_{maxF}}]$.

From here, a new type of variables is introduced, namely the *directly involved variables* (DIV) v_δ which are those variables whose alteration is directly the result of a certain type of AC. Generally, these variables also are part of the feature set, but if not, a relationship must be established between the variables of the feature set and these new variables. These new

variables lead to the definition of the *equivalent directly involved variables* (EDIV) v_ϵ. For simplification, in this study just DIV-type set of variables will be considered for analyzing the ACs. These definitions can be expressed as:

$$\begin{aligned} v_\delta &= \{v_{\delta 1}\ v_{\delta 2}\ ...\ v_{\delta N_\delta}\} \\ v_\epsilon &= \{v_{\epsilon 1}\ v_{\epsilon 2}\ ...\ v_{\epsilon N_\epsilon}\}, v_\epsilon \subset \mathfrak{F} \\ v_E &= \{v_{E1}\ v_{E2}\ ...\ v_{EN_E}\}, v_E \subset \mathfrak{F}_\mathfrak{E} \subset \mathfrak{F} \end{aligned} \tag{4.3}$$

For predicting the post-failure flight envelope, it is necessary to have a concept of the self as described by Perhinschi et al. [42]. The self can simply be viewed as the nominal flight envelope that comes from the normal behaviour of the feature variables. However, if there is any failure or failures $f_i \subset \mathfrak{F}, i = 1, 2, 3\ ...\ N_F$, there are going to be a set of $N_{\Gamma i}$ constraints Γ_i on a set of known variables X_i. These can be expressed mathematically as:

$$\Gamma_i = \{\gamma_{i1}\ \gamma_{i2}\ ...\ \gamma_{i_{N_{\Gamma i}}}\} \tag{4.4}$$

where:

$$\gamma_{ij} = \gamma_{ij}(X_i) = 0, \quad i = 1, 2,\ ...\ N_F, \quad j = 1, 2,\ ...\ N_{\Gamma i} \tag{4.5}$$

Now, assume that the flight envelope alteration can be assessed in terms of a set of Y variables that are also part of the feature set:

$$Y = \{y_1\ y_2\ ...\ y_{N_Y}\} \subset \mathfrak{F} \tag{4.6}$$

Then, based on these variables, the prediction of the flight envelope at post-failure conditions would be equivalent to generating a new self based on the set of features Y. The advantage of this is that the N_Y-dimensional projection can always be obtained and the self-projections of the variables can help to asses the impact of the AC on the nominal flight envelope. Similar as before, the post-failure range of each y_i would be $y_i \in [min(y_i), max(y_i)], i = 1, 2,\ ...\ y_{N_y}$.

4.2 Flight Envelope Reduction for Safe Trajectory Generation

The merging between the RRT* path planner and the Dubins Airplane Paths presented in Chapter 3 requires strictly definitions of the maximum flight path angle limits and roll/bank angle limits to generate smooth and feasible trajectories. These angles are closely related to the aileron and elevator actuator deflections, for which any failure in these control surfaces could be detected, identified, and evaluated within the AIS paradigm. Specifically, this thesis will be focusing on the failure cases related to the fixed jamming of both ailerons with positive and negative fixed deflections. However, note that the concepts presented in the previous sections are applicable, in general, to any type of failures and in other subsystems such as the propulsion plant on the UAV or other control surfaces.

The main goal of this section is to provide a methodology to estimate how the roll capability can be constrained whenever there is some jamming on one of the ailerons. The assumption here is that the flight path angle remains at nominal conditions while maintaining the same coordinate turn conditions. Table 4.1 shows the type of AC evaluation that will be the main focus of the analysis. The DIVs and ERVs variables are shown along with the specific jamming failure cases at certain flight times.

Table 4.1 AC Evaluation Analysis

#	Type of Failure	Deflection Degree of Blockage	DIV	ERV
1	Jammed Right Aileron (High Magnitude)	$15°$ at $t = 18s$	δ_{ail_R} δ_{ail_L}	$p, r, pb/2V, \phi$ $\delta_{rud_L}, \delta_{rud_R}$
2	Jammed Right Aileron (Low Magnitude)	$-10°$ at $t = 18s$		
3	Jammed Left Aileron (High Magnitude)	$17°$ at $t = 30s$		
4	Jammed Left Aileron (Low Magnitude)	$-8°$ at $t = 33s$		

The magnitude of the failure has been classified as a high/low magnitude failure. This will be useful for the AIS identification part to identify more accurately the level of blockage of the aileron such that the multi-goal RRT* can re-plan the path accordingly. With this, the ambiguity that can exist of the aileron failing at low or high magnitudes can be solved.

4.2.1 Roll Angle Constraints

When one of the ailerons is stuck in flight, the lateral flight performance of the aircraft is affected, and can be quantified by two criteria: the post-failure maximum achievable roll angle ϕ_{max}, and the time τ_{roll} the aircraft needs to achieve that roll angle. For simplicity, the time variable will be neglected as it is assumed that the size of the ailerons is large enough to have the necessary control authority to achieve the needed roll angle even if one of them is at failure conditions (this would make τ_{roll} negligible small).

4.2.1.1 Left Aileron Failure

Based on the mentioned idea, Ducard et al. [47] propose the reduction of the lateral flight envelope by considering the roll mode approximation as follows. Let the relative deflection of the ailerons be defined by Equation 4.7.

$$\Delta\delta_{ail} = \delta a_1 - \delta a_2 \tag{4.7}$$

where: δa_1 and δa_2 are the left and right aileron deflections, respectively. During straight-level flight, if one aileron fails, then the controller would have to deflect the other aileron such that $\Delta\delta_{ail} = 0$. First, consider the scenario in which the left aileron is stuck, as depicted in Figure 4.1, where it can also be seen that the right aileron tries to compensate for the jammed deflection.

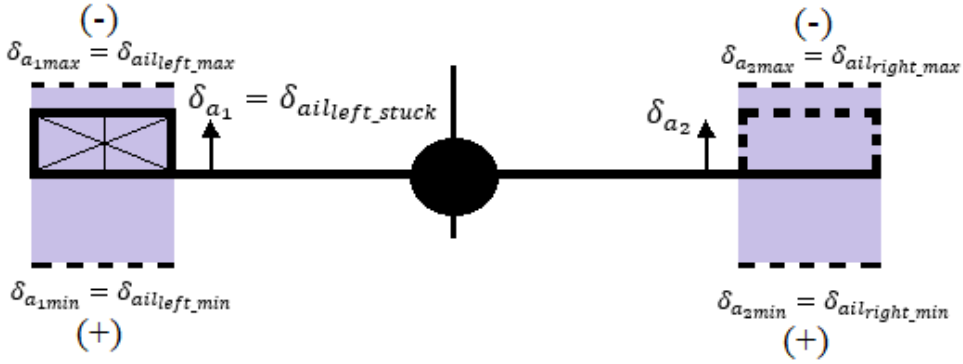


Figure 4.1 Aircraft Rear View: Left Aileron Stuck.

Here, the maximum deflections are shown with the upward deflections being negative, and downward deflections being positive as previously mentioned. This configuration allows to define the maximum relative deflection for the right and left bank as follows:

$$\begin{aligned}(\Delta\delta_{ail_{right\,bank}})_{max} &= \delta a_{1_{stuck}} - \delta a_{2_{max}} \\ (\Delta\delta_{ail_{left\,bank}})_{max} &= \delta a_{1_{stuck}} - \delta a_{2_{min}}\end{aligned} \tag{4.8}$$

After a time interval Δt, the maximum achievable bank angles in terms of the roll rate p in both directions would be:

$$\begin{aligned}\phi_{left} &= -|p_{left_{max}}|\Delta t \\ \phi_{right} &= -|p_{right_{max}}|\Delta t\end{aligned} \tag{4.9}$$

Since the general stability of the aircraft must be preserved, then it would be required that the left bank angle has a smaller value so that it takes the same time for the aircraft to come back from $\phi_{left_{max}}$ as from $\phi_{right_{max}}$ to maintain $\phi = 0°$. These can be translated mathematically:

$$\frac{\phi_{left_{max}}}{|p_{right_{max}}|} = -\frac{\phi_{right_{max}}}{|p_{left_{max}}|} \tag{4.10}$$

Now, recall from the general dynamics of an aircraft that the equation for the angular rates (further used in Chapter 5 for the derivations of the equations of motion) can be expressed in terms of the moments around each axis as:

$$\dot{\omega} = I^{-1}([L, M, N]' - \omega \times I\omega) \tag{4.11}$$

where: $\omega = [p, q, r]'$ and I is the inertia matrix of the vehicle as further defined in Equation 5.7. For the reminder of this study, it is assumed that the total roll, pitch, and yaw moments are the sum of the moments produced by the aerodynamic effects and the control surface actuators at the same time, that is:

$$[L, M, N]' = [L, M, N]'_{aero} + [L, M, N]'_{surf} \tag{4.12}$$

Recall also that the rolling moment L can be expressed in the form of Taylor series as:

$$L = \bar{q}Sb(C_{l_{\delta_a}}\Delta\delta_{ail} + C_{l_p}p + C_{l_{\delta_r}}r + C_{l_\beta}\beta) \tag{4.13}$$

where the lateral derivatives $C_{l_{\delta_a}}, C_{l_p}, C_{l_{\delta_r}}, C_{l_\beta}$ are defined in Table 5.3 for the Rascal 110. Then, a roll mode approximation can be obtained of the form of Equation 4.14 where the air density ρ is assumed to be constant at sea level:

$$\dot{p} = \frac{\rho V I_{zz} S b^2 C_{l_p}}{4(I_{xx}I_{zz} - I_{xz}^2)}p + \frac{\rho V^2 I_{zz} S b C_{l_{\delta_\alpha}}}{2(I_{xx}I_{zz} - I_{xz}^2)}\Delta\delta_{ail} \tag{4.14}$$

where: V is the aircraft true airspeed. For simplification, define two positive constants θ_p and θ_{ail} as:

$$\begin{aligned} \theta_p &= -\frac{\rho I_{zz} S b^2 C_{l_p}}{4(I_{xx}I_{zz} - I_{xz}^2)} \\ \theta_{ail} &= \frac{\rho I_{zz} S b C_{l_{\delta_\alpha}}}{2(I_{xx}I_{zz} - I_{xz}^2)} \end{aligned} \tag{4.15}$$

Then, Equation 4.14 becomes:

$$\dot{p} = -V\theta_p p + V^2\theta_{ail}\Delta\delta_{ail} \tag{4.16}$$

This differential equation can be integrated with the assumption that the velocity is almost constant or that it is changing slowly. The result is the known roll rate time response:

$$p = V\frac{\theta_{ail}}{\theta_p}\Delta\delta_{ail}(1 - e^{-V\theta_p t}) \tag{4.17}$$

Equation 4.17 implies that the ratio between the left and right roll rates is proportional to the maximum relative deflections of the ailerons in each direction. Equation 4.18 shows this:

$$\frac{|p_{left\,max|}}{|p_{right\,max|}} = \frac{(\Delta\delta_{ail_{left\,bank}})_{max}}{(\Delta\delta_{ail_{right\,bank}})_{max}} = \mathfrak{R}_{p_{max}} \tag{4.18}$$

Finally, the maximum left bank angle at post-failure can be computed by assuming that the aircraft can still bank nominally to the right, as shown in Equations 4.19:

$$\begin{aligned} \phi_{right\,max} &= \phi_{max,nominal} = +50^{\circ} \\ \phi_{left\,max} &= \frac{-\phi_{right\,max}}{\mathfrak{R}_{p_{max}}} \end{aligned} \tag{4.19}$$

4.2.1.2 Right Aileron Failure

Similar as for the left aileron, expressions for the stuck right aileron can also be derived. Consider Figure 4.2 which is the right aileron stuck at a positive deflection.

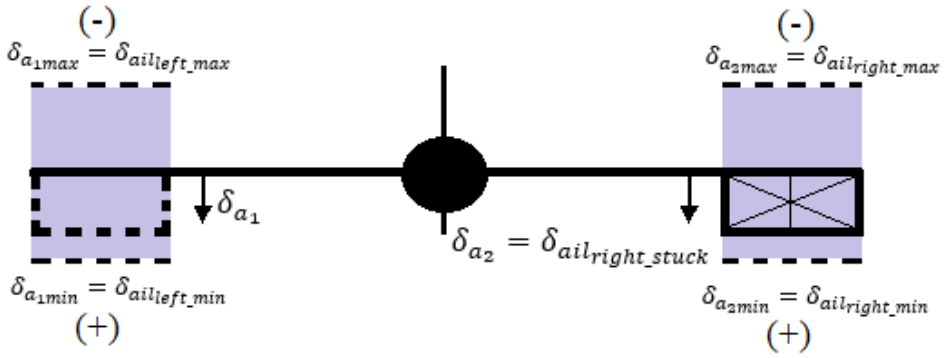


Figure 4.2 Aircraft Rear View: Right Aileron Stuck.

Then, the maximum relative deflection angles for both directions can be derived to be:

$$\begin{aligned} (\Delta\delta_{ail_{right\,bank}})_{max} &= \delta a_{1_{min}} - \delta a_{2_{stuck}} \\ (\Delta\delta_{ail_{left\,bank}})_{max} &= \delta a_{1_{max}} - \delta a_{2_{stuck}} \end{aligned} \tag{4.20}$$

Following the same procedure as before, and assuming a relatively constant-slow changing velocity, then Equations 4.21 can be obtained:

$$\begin{aligned} \phi_{left\,max} &= \phi_{max,nominal} = -50^{\circ} \\ \phi_{right\,max} &= -\frac{\phi_{left\,max}}{\mathfrak{R}_{p_{max}}} \end{aligned} \tag{4.21}$$

Notice that these derivations can be classified just as approximations since Equation 4.17 strictly shows a first-order exponential solution. However, for the purposes of this study, these approximations have proven to be satisfactory enough to demonstrate how the roll angle is restricted, thereby affecting the entire flight envelope at post-failure conditions.

4.3 Antibody Generation Algorithm

The AIS paradigm requires the generation of the self and the creation of antibodies in order to be able to detect and potentially identify certain types of failures. For doing this, the very first step of the process is to select adequate feature variables that will depend on the type of failure in analysis and on the system/subsystem where the abnormal condition occurs.

Once the set of features is selected, then several flight tests in simulation or real-world must be conducted to record data of these features. These data will then go to a normalization and clustering process for creating the hyperspace of the self through an off-line training process that will result in the creation of N-dimensional projections. Based on the dimensionality of this hyperspace, then a set of detectors will be created in order to fill the rest of the empty hyperspace based on a number of requirements (further described in subsection 4.3.2) such that, through the use of a NS algorithm, abnormal flight data points can be detected whenever they fall into the detectors zone. Figure 4.3 highlights the main steps of the mentioned process as described by Perhinschi and Moncayo [48], also known as the Raw Data set Union Method (RSDUM). Here, after the flight is conducted and data has been recorded, a single raw data file is obtained where it is further processed to generate the desired projections.

Due to the computational requirements and for visualization purposes, the selected projections are $N = 2$ - dimensional as stated in subsection 1.1.5. The generated projections will be based on the features described by Table 4.2 for the aileron failure cases. As can be seen, the features for the ailerons analysis are mainly focused on the lateral dynamics of the aircraft as this type of motion is putting constraints on the overall flight envelope.

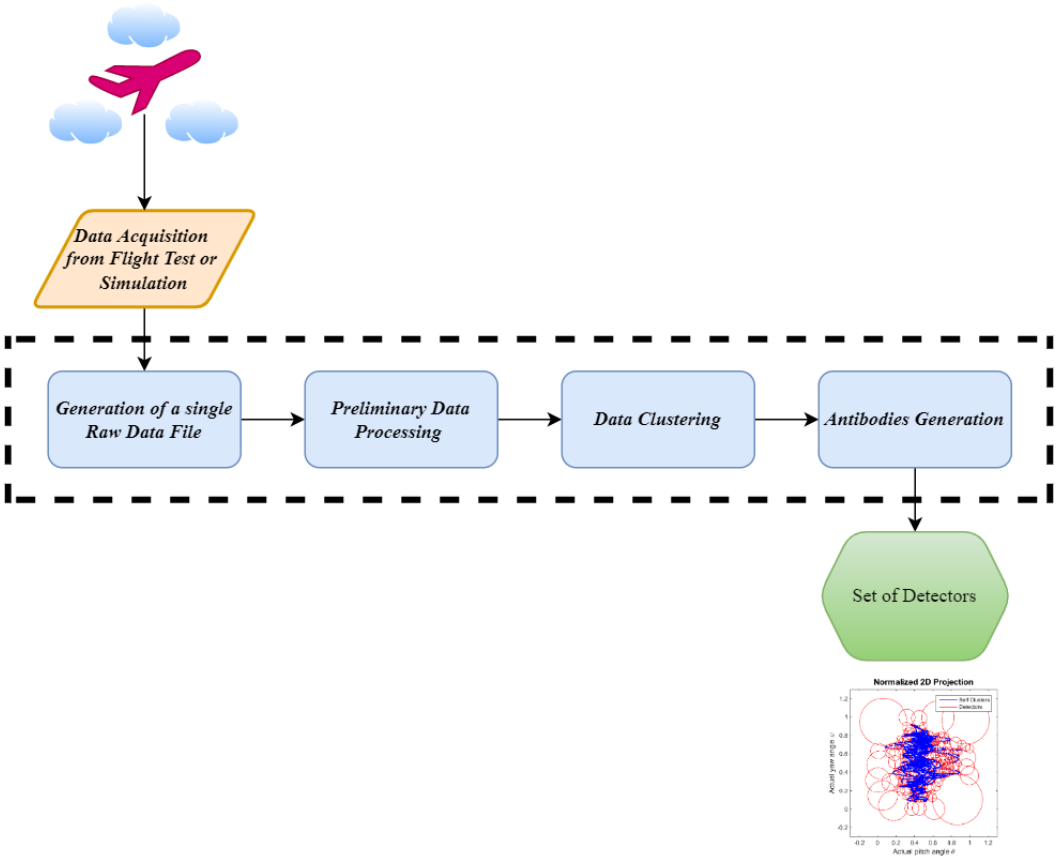


Figure 4.3 Antibody Generation Process based on RSDUM.

Table 4.2 Selected features for AIS Application for Aileron failure cases.

#	**Selected Features for Lateral Dynamics**	
1	p	Roll Rate
2	r	Yaw Rate
3	$pb/2V$	Helix Angle or Normalized Roll Rate
4	ϕ	Roll Angle
5	δ_{ail_L}	Left Aileron Deflection
6	δ_{ail_R}	Right Aileron Deflection
7	δ_{rud_L}	Left Rudder Deflection
8	δ_{rud_R}	Right Rudder Deflection

Theoretically, there are a certain number of possible two-dimensional combinations N_{proj} that the projections could have. This number can be calculated using the binomial coefficient in Equation 4.22 for each failure case. The number of features $N_{feat_{ail}} = 8$ is used.

$$N_{proj_{ail}} = C_N^{N_{feat_{ail}}} = \frac{N_{feat_{ail}}!}{N!(N_{feat_{ail}} - N)!} = \frac{8!}{2!(8-2)!} = 28 \tag{4.22}$$

Notice, however, that not all the projections will show selves that are useful to detect and identify the failure since some of them will not provide enough information due to its similarity to the nominal selves obtained when no AC is present. Therefore, just sets of $N_{proj_{ail}} = 10$ projections were selected as a subset of all the available projections. These combination of features for producing the desired selves are shown in Table 4.3.

Table 4.3 Selected Projections for Detection and Identification Scheme.

#Self	Projections for Aileron Failure
1	p, δ_{ail_L}
2	p, δ_{ail_R}
3	$pb/2V, \delta_{ail_L}$
4	$pb/2V, \delta_{ail_R}$
5	r, δ_{ail_L}
6	r, δ_{ail_R}
7	δ_{ail_L}, ϕ
8	δ_{ail_R}, ϕ
9	$\delta_{rud_L}, \delta_{ail_R}$
10	$\delta_{rud_R}, \delta_{ail_L}$

The results of the application of this methodology are further shown in Chapter 6 along with the application of some performance metrics that are useful to evaluate the feasibility of the new generated trajectories.

4.3.1 Single-Data File Generation

The flight simulation tests for this thesis record data of about 42 state variables at a rate of 50 Hz that are rearranged into a matrix of 42 columns. In a real-world application, normally these data would be saved on the SD card of a microcontroller or autopilot hardware. First, data from the nominal flight following a specific trajectory is saved for obtaining the nominal self trough a supervised learning approach. Then, new flight data with failures can be recorded for further analysis.

4.3.2 Data Preprocessing and Clustering

Once the single-data file is obtained, then the data processing phase begins. Naturally, this file contains features whose magnitudes change dramatically in comparison with each

other, and it would be larger in size even if a higher recording frequency is desired. Due to this, and for the purpose of reducing the file size, a normalization procedure is applied which makes all the variables stand between 0 and 1. Additionally, duplicate points are deleted to obtain a more compact file and save processing time.

After normalization, an algorithm based on the "k-means" is used to cluster the data points of each feature. Perhinschi and Moncayo [48] use this as a method for grouping data to be used in the projections. Notice that for this thesis, the clusters are represented as circles, though they can be N-dimensional and not necessarily suitable to be visualized. These circles are often referred to as hyper-bodies or hyper-spheres that are used to fill the hyperspace. The hyper-spheres need to comply with certain characteristics in order to be suited well into the hyperspace. Garcia [43] describes these design characteristics, as shown in Table 4.4.

Table 4.4 Criteria for Designing Hyper-Spheres.

Characteristics of hyper-bodies
Minimum overlapping in-between the self clusters
Minimum overlapping in-between the antibodies
No overlapping between the self and the antibodies
Minimum uncovered areas in the antibodies (non-self)
Minimum number of detectors
Minimum empty space in the self clusters

Finally, the NS algorithm needs additional parameters that the user must specify such as the number of initial detectors, the maximum number of allowed iterations, the minimum radius permitted for a hyper-body, and the number of clusters to be used. The generation of the hyper-bodies will stop as soon as the maximum number of iterations is reached, or as the hyper-space is completely explored. These parameters can be tuned according to the application, and they are further shown in Chapter 6.

4.3.3 AIS Detection Metrics

Due to the nature of the AIS algorithm, there exists the possibility that false detection comes into play due to sensor noise or other factors that can trigger false alarms of failure detection. There also exists the possibility that no detection is performed even if a failure is actually compromising the system. Therefore, a performance evaluation of the entire HMS is needed, and this is based on the Detection Rate (DR) and False Alarms (FA) calculated through Equation 4.23 and Equation 4.24, respectively.

$$DR = \frac{TP}{TP + FN} \times 100 \tag{4.23}$$

$$FA = \frac{FP}{TN + FP} \times 100 \tag{4.24}$$

where: True Positive (TP) refers when a failure is detected and declared as failure, True Negative (TN) refers when nominal conditions are declared as nominal, False Positive (FP) refers when nominal conditions are declared as failure, and False Negative (FN) refers when the failure is not detected at all. Of course, the performances pointed out by these metrics highly depend on the features selected to analyze each specific failure case. The main goal is to diminish the false alarms while improving the detection rate for all cases.

4.3.4 Detection and Identification Scheme

The Detection and Identification algorithms are embedded into the entire HMS block in Simulink shown in Figure 4.4. The input needed for this block is the discrete signal that contains the simulated sensor readings and parameters of the 42 variables that are being recorded during the entire flight. In order to detect any type of failure first, a window of $w = 10$ samples is selected and two thresholds are defined in terms of the number of activated detectors (NAD), such that if those samples instantaneously surpass the first threshold, a warning of a failure will be displayed, and if they surpass the second threshold, a definite failure detection will appear which will activate the replanning logic of the RRT* multi-goal path planner. The type of failure will appear on the display according to its magnitude.

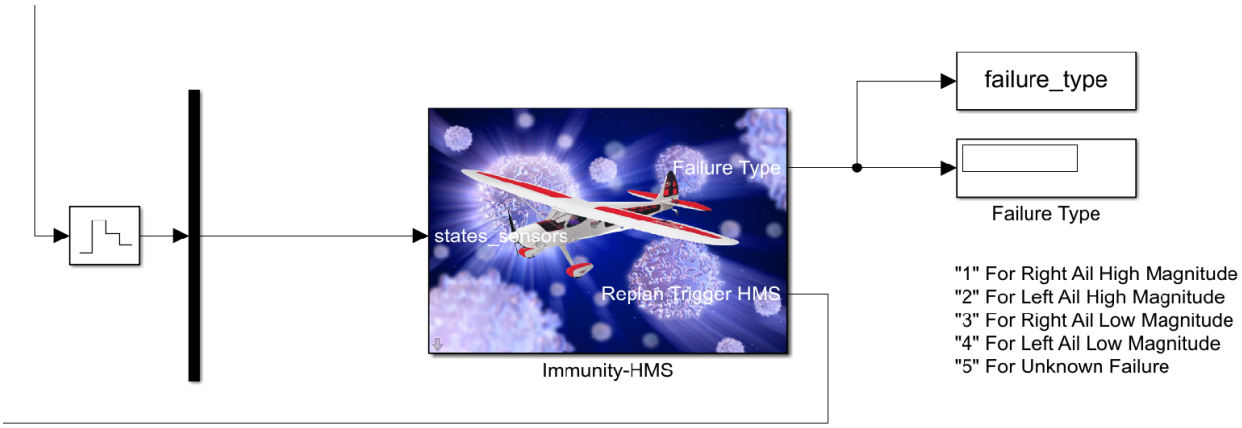


Figure 4.4 HMS(AIS) Simulink block.

The NAD and the type of failure come from the Detection and Identification subsystems shown in Figure 4.5, respectively. Notice that a conditional is set after the detection display block which basically makes a signal to be triggered once threshold 2 (Thr2) is exceeded to activate the replanning mission. A data type conversion block is used to change the variable to be a double instead of a Boolean.

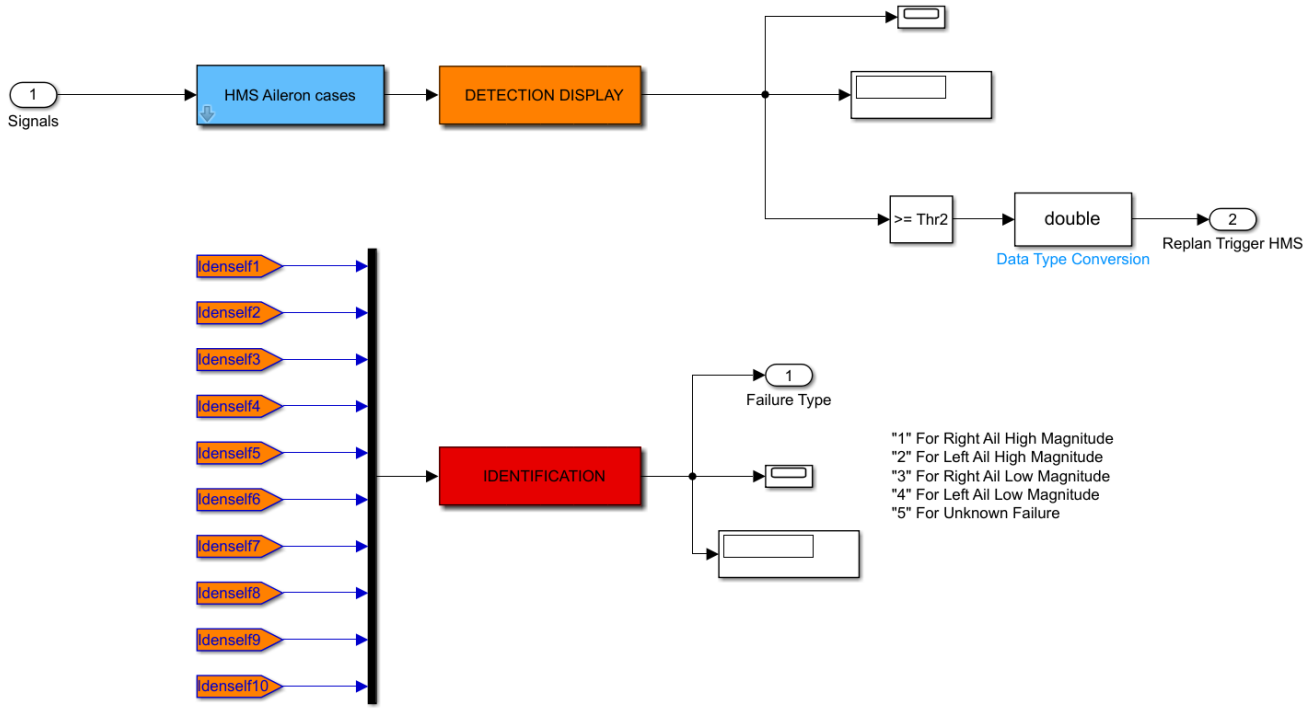


Figure 4.5 Detection and Identification Subsystems.

Now, the NAD are calculated and buffered in the ten selves which are defined as S-functions inside the "HMS Aileron cases" block. As an example, Figure 4.6 shows this only for selves $5, 6, 7$. These selves represent the UAV dynamic fingerprint within its nominal flight condition. In this way, while the fixed-wing is on flight, the nominal selves are instantaneously compared with the data received, and if the data points falls into the detectors zone due to a failure, then the number of the activated detectors is recorded. At the end, every NAD of every self is added for activating the overall detection logic.

Finally, the selves S-Functions also implement a logic whose main task is to identify which specific detectors are being activated within each type of failure so the failures can be further classified according to their type and magnitude level. For achieving this, each failure is inserted separately and then the user can manually record the detectors that are mostly being activated during the post-failure condition. The main advantage of doing this training procedure is that the HMS becomes robust enough to detect and identify the failure. However, its main disadvantage is that it is still a supervised training process.

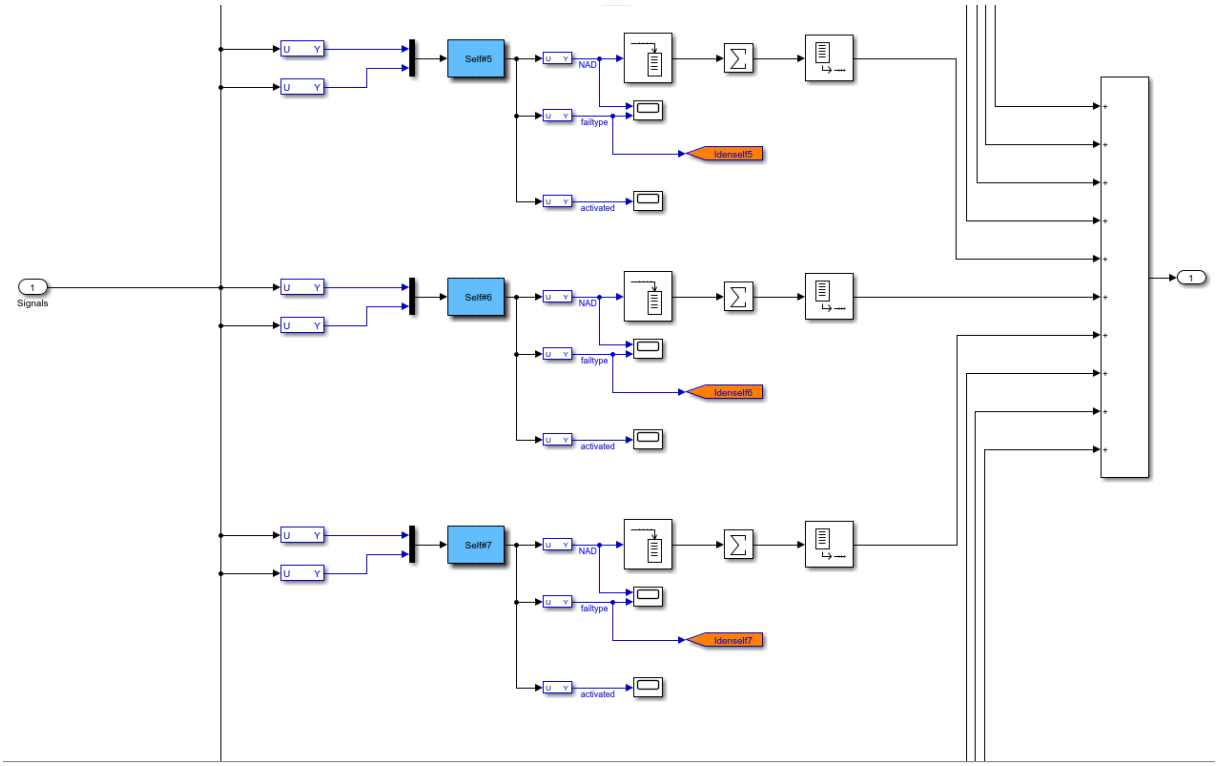


Figure 4.6 Selves S-Functions.

5 Simulation Environment

5.1 Rascal 110: Physical and Dynamical Parameters

The fixed-wing UAV model used in this study is based on a well know aircraft called Rascal 110. Previous studies have been conducted at the Advanced Dynamics and Control Lab at ERAU that have completely quantified the physical, dynamical, and geometrical parameters of this model.

First, to obtain the stability derivatives, a mesh model of the aircraft was generated in order to perform a Computational Fluid Dynamics (CFD) analysis through the use of Digital Datcom. This program was developed by NASA and the USA Air Force, and it provides a cost-effective approach to estimate the longitudinal and lateral derivatives of conventional aircraft through the subsonic regime.

The initial study was performed by Lyons [49], and has been recurrently used for performing other types of analyses such as system identification and development of non-linear adaptive controls laws [4],[50]. Figure 5.1 [49],[4] shows a comparison between the generated 3D mesh model and the original model of the aircraft, and it also shows the real balsa wood model assembled at the Advanced Dynamics and Control Lab at ERAU.

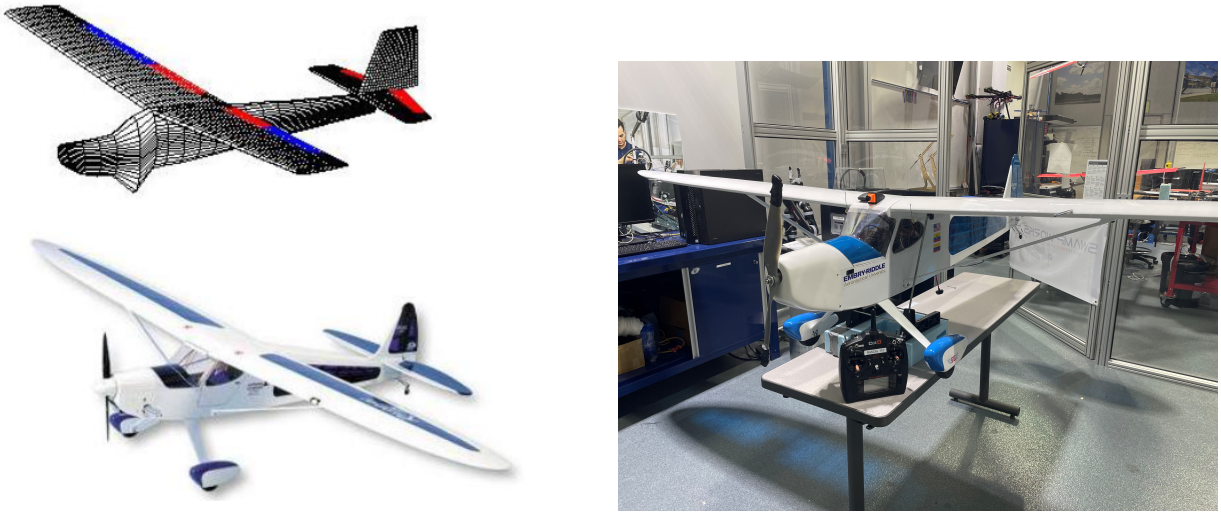


Figure 5.1 Rascal 110 Mesh Model and Original Model (left) vs. Assembled Balsa Wood Model (right).

The most important physical parameters of the model are shown in Table 5.1. Additionally, Table 5.2 [49] shows the input parameters that were introduced into Digital Datcom to perform the analysis. Table 5.3 shows the results of the study, where the longitudinal and lateral derivatives of the UAV are shown. The quantification of all these variables is essential as they mark the dynamical fingerprint of the vehicle and are needed for the equations of motion described in the next section.

Table 5.1 Rascal 110 Physical Parameters.

Physical Parameter	**Symbol**	**Value**	**Units**
Wingspan	b	2.80	m
Wing Planform Area	S	0.98	m
Mean Aerodynamic Chord	$\bar{c}$	0.35	m
Mass	m	7	kg
Inertia about X-axis	I_{xx}	2.64	$kg \cdot m^2$
Inertia about Y-axis	I_{yy}	2.10	$kg \cdot m^2$
Inertia about Z-axis	I_{zz}	2.59	$kg \cdot m^2$

Table 5.2 Input Parameters for Datcom analysis.

Parameter	**Symbol**	**Value**	**Units**
Speed	V	68.10	ft/s
Altitude	h	0	ft
Chord	c	1.25	ft
Wing Area	S	10.57	ft^2
Span	b	9.17	ft
Weight	W	15.74	lb

Lastly, Table 5.4 describes the maximum deflections that the main control surfaces of the aircraft are allowed to have during the course of the commanded trajectory. The actuator models are based on a first order transfer function as described by Equation 5.1, which represents a delay of 0.1s.[50]

$$G(s) = \frac{20}{s+30} e^{-0.1s} \tag{5.1}$$

Table 5.3 Rascal 110 Stability Derivatives.

Longitudinal Derivatives		**Units**	**Lateral Derivatives**		**Units**
C_{L_0}	0.4940	per rad	C_{l_β}	-0.1002	per rad
C_{L_α}	5.973		C_{l_p}	-0.5087	
C_{L_q}	4.885		$C_{l_{\delta_a}}$	0.4698	
$C_{L_{\delta_e}}$	0.200		$C_{l_{\delta_r}}$	0.0103	
C_{D_0}	0.031		C_{n_β}	0.01274	
C_{D_α}	0.527		C_{n_p}	-0.03802	
$C_{D_{\delta_e}}$	0.000		C_{n_r}	-0.03777	
C_{M_0}	0.0323		$C_{n_{\delta_a}}$	-0.00190	
C_{L_α}	-0.3217		$C_{n_{\delta_r}}$	-0.12200	
C_{M_q}	-11.000		C_{Y_β}	-0.3198	
$C_{M_{\delta_e}}$	-0.5517		C_{Y_p}	-0.1138	

Table 5.4 Allowable Control Surface Deflections.

Control Surface	**Max. and Min. Deflection**
Aileron	$\delta_{a_{max,min}} = \pm 27\,\text{deg}$
Elevator	$\delta_{e_{max,min}} = \pm 27\,\text{deg}$
Rudder	$\delta_{r_{max,min}} = \pm 35\,\text{deg}$

5.2 Equations of Motion

In order to build the simulation, a six-degree-of-freedom (6-DoF) model of the UAV must be integrated to Simulink. This is done by considering the 12 non-linear equations of motions (EOMs) that describe, in general, the motion of any conventional aircraft. These EOMs fall under four categories as follows:

1. Force Equations for deriving linear accelerations (i.e $[\dot{u}, \dot{v}, \dot{w}]$).

2. Moment Equations for deriving the rate of change of angular rates (i.e $[\dot{p}, \dot{q}, \dot{r}]$).

3. Kinematic Equations for deriving Euler Rates (i.e $[\dot{\phi}, \dot{\theta}, \dot{\psi}]$).

4. Navigation Equations for deriving velocities in the Earth Reference Frame (i.e $[\dot{X}, \dot{Y}, \dot{Z}]$).

Nelson [51] shows a complete derivation of these equations. However, for the purposes of this work, just the most important expressions are shown.

Assuming that the Earth Reference Frame or North-East-Down (NED) Reference Frame is an inertial frame relative to the Aircraft-Body-Centered (ABC) Reference Frame, the first task is to obtain a Rotation Matrix that allows an easy conversion between both frames. This matrix is well known as the Direction Cosine Matrix (DCM), following a 3-2-1 sequence or a "yaw-pitch-roll" sequence. The main goal for this is to express the velocity of the aircraft in the NED frame in terms of the Euler angles $[\phi, \theta, \psi]$ and body velocities components $[u, v, w]$. Particularly, this will be useful to derive the Navigation Equations. Equation 5.2 shows the DCM which is the result of the multiplication between the rotation around each 3D axis.

$$R_B^E = \begin{bmatrix} cos(\theta)cos(\psi) & -cos(\phi)sin(\psi) + sin(\phi)sin(\theta)cos(\psi) & sin(\phi)sin(\psi) + cos(\phi)sin(\theta)cos(\psi) \\ cos(\theta)sin(\psi) & cos(\phi)cos(\psi) + sin(\phi)sin(\theta)sin(\psi) & -sin(\phi)cos(\psi) + cos(\phi)sin(\theta)sin(\phi) \\ -sin(\theta) & sin(\phi)cos(\theta) & cos(\phi)cos(\theta) \end{bmatrix} \quad (5.2)$$

Where R_B^E means a transformation from ABC to NED.

Figure 5.2 shows the mentioned reference frames and the convention of signs that will be used to derive the equations. Though not shown on this Figure, the convention that is adopted for the control surfaces deflection will consider that a positive aileron deflection $+\delta_a$ and a positive elevator deflection $+\delta_e$ move the control surfaces downwards, and vice versa.

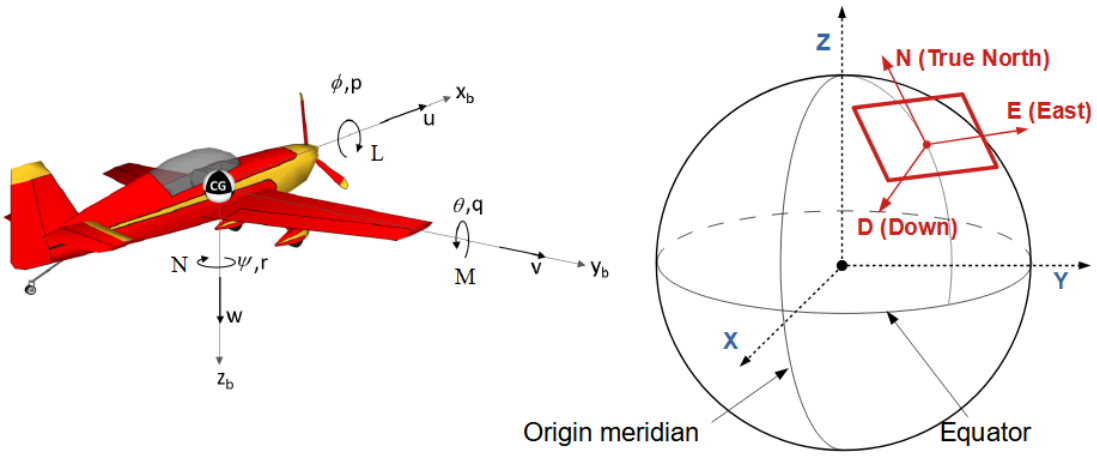


Figure 5.2 ABC (left) and NED (right) Reference Frames.

Force Equations

By applying Newton's Second Law shown in Equation 5.3, and assuming a constant mass, the set of equations in 5.4 are obtained.

$$\sum \vec{F}_{ext} = m \left. \frac{d\vec{v}}{dt} \right|_I \tag{5.3}$$

where: $\sum \vec{F}_{ext}$ is the sum of all the external forces of the aircraft acting on each axis, m is the mass, and $\left. \frac{d\vec{v}}{dt} \right|_I$ is the rate of change of the velocity in the inertial frame.

$$\frac{\sum \vec{F}_{ext}}{m} = \begin{bmatrix} \dot{u} + qw - rv \\ \dot{v} + ru - pw \\ \dot{w} + pv - qu \end{bmatrix} \tag{5.4}$$

Now, the linear accelerations can be solved in the ABC reference frame as shown in Equations 5.5. These are the first three Equations needed.

$$\begin{bmatrix} \dot{u} \\ \dot{v} \\ \dot{w} \end{bmatrix} = \begin{bmatrix} \frac{F_x}{m} - qw + rv \\ \frac{F_y}{m} - ru + pw \\ \frac{F_z}{m} - pv + qu \end{bmatrix} \tag{5.5}$$

Moment Equations

For the Moment Equations, Newton's Second Law is applied to describe the rotational motion of the body (also known as angular momentum). For this, the momentum of momentum is used along with the Coriolis term, described in Equation 5.6.

$$\sum \vec{M}_{ext} = m\frac{d\vec{H}_B}{dt} + (\vec{\omega} \times \vec{H}_B) \tag{5.6}$$

where: $\vec{H}_B = \vec{\omega} \times I$ is the momentum of momentum, $\vec{\omega}$ is the vector of angular velocities, and I is the vehicle's mass moment of inertia.

Since the vehicle is symmetric about the XZ plane, then $I_{xy} = I_{yz} = 0$, and the matrix of the inertia gets simplified as follows.

$$I = \begin{bmatrix} I_{xx} & -I_{xy} & -I_{xz} \\ -I_{xy} & I_{yy} & -I_{yz} \\ -I_{xz} & -I_{yz} & I_{zz} \end{bmatrix} = \begin{bmatrix} I_{xx} & 0 & -I_{xz} \\ 0 & I_{yy} & 0 \\ -I_{xz} & 0 & I_{zz} \end{bmatrix} \tag{5.7}$$

Expanding the terms in Equation 5.6 yields the Moments in terms of inertias and angular rates.

$$\begin{bmatrix} L \\ M \\ N \end{bmatrix} = \begin{bmatrix} \dot{p}I_{xx} + qr(I_{zz} - I_{yy} - (\dot{r} + pq)I_{xz} \\ \dot{q}I_{yy} - pr(I_{zz} - I_{xx}) + (p^2 - r^2)I_{xz} \\ \dot{r}I_{zz} + pq(I_{yy} - I_{xx} + (qr - \dot{p}I_{xz} \end{bmatrix} \tag{5.8}$$

where: $[L, M, N]'$ is the vector of moments around the X, Y, Z axes respectively.

Given the Equations in 5.8, the next three expressions needed for the rate of change in the angular rates can be obtained:

$$\begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} = - \begin{bmatrix} \frac{qr(I_{zz} - I_{yy})}{I_{xx}} \\ \frac{pr(I_{xx} - I_{zz})}{I_{yy}} \\ \frac{pq(I_{yy} - I_{xx})}{I_{zz}} \end{bmatrix} + \begin{bmatrix} \frac{1}{I_{xx}} & 0 & 0 \\ 0 & \frac{1}{I_{yy}} & 0 \\ 0 & 0 & \frac{1}{I_{zz}} \end{bmatrix} \begin{bmatrix} L \\ M \\ N \end{bmatrix} \tag{5.9}$$

Kinematic Equations

By performing transformations using the Euler angles and the angular rates, the Kinematic Equations are derived, which represent the rate of change in the Euler angles. Notice that, in general, $[p, q, r]' \neq [\phi, \theta, \psi]'$. The final expression are shown in Equations 5.10.

$$\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} 1 & sin(\phi)tan(\theta) & cos(\phi)tan(\theta) \\ 0 & cos(\phi) & -sin(\phi) \\ 0 & sin(\phi)sec(\theta) & cos(\phi)sec(\theta) \end{bmatrix} \begin{bmatrix} p \\ q \\ r \end{bmatrix} \tag{5.10}$$

Navigation Equations

Finally, the Navigation Equations are obtained by using the DCM mentioned before to transform the body linear velocities into the NED frame. As such, Equation 5.12 is applied.

$$\begin{bmatrix} \dot{X} \\ \dot{Y} \\ \dot{Z} \end{bmatrix} = R_B^E \begin{bmatrix} u \\ v \\ w \end{bmatrix} \tag{5.11}$$

Expanding this expression yields to the last three EOMs:

$$\begin{bmatrix} \dot{X} \\ \dot{Y} \\ \dot{Z} \end{bmatrix} = R_B^E = \begin{bmatrix} cos(\theta)cos(\psi) & -cos(\phi)sin(\psi) + sin(\phi)sin(\theta)cos(\psi) & sin(\phi)sin(\psi) + cos(\phi)sin(\theta)cos(\psi) \\ cos(\theta)sin(\psi) & cos(\phi)cos(\psi) + sin(\phi)sin(\theta)sin(\psi) & -sin(\phi)cos(\psi) + cos(\phi)sin(\theta)sin(\phi) \\ -sin(\theta) & sin(\phi)cos(\theta) & cos(\phi)cos(\theta) \end{bmatrix} \begin{bmatrix} u \\ v \\ w \end{bmatrix} \tag{5.12}$$

For completing all the necessary variables, the true airspeed V_{TAS}, angle of attack α, and sideslip angle β are computed as follows:

$$V_{TAS} = \sqrt{u_r^2 + v_r^2 + w_r^2} \tag{5.13}$$

$$\alpha = tan^{-1}\left(\frac{w_r}{u_r}\right) \tag{5.14}$$

$$\beta = sin^{-1}\left(\frac{v_r}{\sqrt{u_r^2 + v_r^2 + w_r^2}}\right) \tag{5.15}$$

where the velocity of the wind is taken into account in the following expressions:

$$\begin{aligned} u_r &= u + u_{wind} \\ v_r &= v + v_{wind} \\ w_r &= w + w_{wind} \end{aligned} \tag{5.16}$$

In summary, the set of Equations in 5.5,5.9,5.10,5.12, along with Equations 5.13,5.14, and

5.16 describe the full motion and orientation of the vehicle, and they are integrated into the Simulink environment for the simulation cases.

5.3 Tracking the Safe Trajectory

In order to make it possible for the UAV to track the generated trajectory by the RRT* path planner, a modified version of a Formation Geometry Calculation (FGC) is implemented in junction with a Nonlinear Dynamic Inversion (NLDI) Controller. The following subsections explain these two approaches in detail, which together compose the entire Formation Flight Control (FFC).

5.3.1 Formation Flight Control: Virtual Trajectory Tracking

Campa et al.[52] provide a detail description of Formation Control Laws. The concept relies on having a leader aircraft being followed autonomously by a follower aircraft (often called the wingman), while the former is being commanded manually (i.e. by an RC control). However, this concept can be slightly modified by replacing the leader aircraft with the desired trajectory to be tracked. In this way, the "virtual trajectory tracking" term acquires its name and the leader aircraft becomes a virtual leader.

Consider Figure 5.3 [52] which shows the formation geometry in analysis on the NED frame. According to the authors, the problem can be subdivided into a horizontal and a vertical tracking problem. Here, f and l represent the forward and lateral errors respectively, while f_c and l_c represent the forward and lateral clearance to maintain a safe distance from the other aircraft. Since the main focus is only to command the wingman to follow the desired trajectory without any other aircraft present, then the f_c and l_c terms are neglected. Additionally, Ω_L represents the induced angular velocity of the virtual leader.

Based on this, Equation 5.17 and Equation 5.18 can be derived where V_{V_y} and V_{V_x} are the the velocity of the virtual leader in each axis, x_v and y_v describe the position of the virtual leader, x_w and y_w is the position of the wingman aircraft (which in this study would be the Rascal 110), and $V_{V_{xy}}$ is the projection of the leader's velocity onto the x-y plane defined as $V_{V_{xy}} = \sqrt{V_{V_x}^2 + V_{V_y}^2}$.

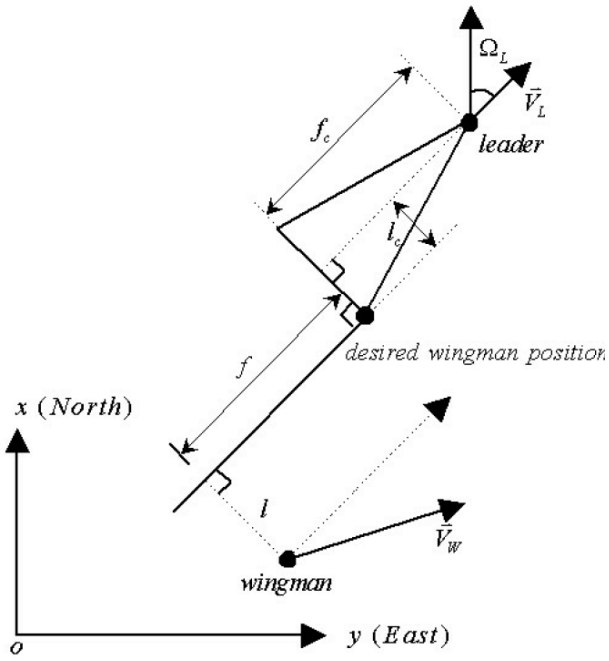


Figure 5.3 Level Plane Formation Geometry.

$$l = \frac{V_{V_y}(x_v - x_w) - V_{V_x}(y_v - y_w)}{V_{V_{xy}}} \tag{5.17}$$

$$f = \frac{V_{V_y}(y_v - y_w) - V_{V_y}(x_v - x_w)}{V_{V_{xy}}} \tag{5.18}$$

For the vertical error h, just the z coordinate of each aircraft is used.

$$h = z_v - z_w \tag{5.19}$$

By introducing the following terms shown in Equations 5.20, the three previous equations can be expressed on a matrix form as shown in the set of Equations 5.21.

$$cos(\chi_v) = \frac{V_{Vx}}{\sqrt{V_{V_x}^2 + V_{V_y}^2}} \qquad\qquad sin(\chi_v) = \frac{V_{Vy}}{\sqrt{V_{V_x}^2 + V_{V_y}^2}} \tag{5.20}$$

$$\begin{bmatrix} l \\ f \\ h \end{bmatrix} = \begin{bmatrix} sin(\chi_v) & -cos(\chi_v) & 0 \\ cos(\chi_v) & sin(\chi_v) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_v - x_w \\ y_v - y_w \\ z_v - z_w \end{bmatrix} \tag{5.21}$$

For computing the velocities, the time derivative is taken from Equations 5.21. An important consideration is that the induced angular velocity of the virtual leader Ω_V is considered to be zero, as the main goal is to track the generated trajectory with no errors. The final result is as follows.

$$\begin{bmatrix} \dot{l} \\ \dot{f} \\ \dot{h} \end{bmatrix} = \begin{bmatrix} V_{w_{xy}} sin(\chi_w - \chi_V) \\ V_{V_{xy}} - V_{w_{xy}} cos(\chi_w - \chi_V) \\ V_{V_z} - V_{w_z} \end{bmatrix} + \Omega_V \begin{bmatrix} f \\ -l \\ 0 \end{bmatrix} \tag{5.22}$$

The above equations are useful to apply the NLDI controller presented in the next subsection, as they represent the first stage of the FFC.

5.3.2 Nonlinear Dynamic Inversion Controller

NLDI controller architecture is used in this thesis due to its ability to counteract the nonlinearities of the system. Given the common equation to represent a nonlinear system given in 5.23, where $f(x)$ are the nonlinear states and $g(x)$ are the control functions, the inversion is only possible if $g^{-1}(x)$ exists. By replacing the derivatives of the states with the desired states, $\dot{x}_{des} = x$, then the controller has the ability to provide direct inputs into the system as shown in Equation 5.24. This is the goal of the NLDI, as direct input commands will be provided into the control surfaces and the propulsion system.

$$\dot{x} = f(x) + g(x)u \tag{5.23}$$

$$u = g^{-1}(x)[\dot{x}_{des} - g(x)] \tag{5.24}$$

Figure 5.4 highlights the controller architecture that is implemented for the simulation. The FFC controller is composed by three sub-controllers, starting from the computation of the lateral, forward and vertical distances, and their respective velocities that where described in the previous subsection. Now, the NLDI controller will serve to build the outer-loop and inner-loop, such that the commanded path can be transformed into actuator and throttle commands. For convenience, just the most important derivations will be shown. More on this topic can be found on References [49],[52],[50], and [53].

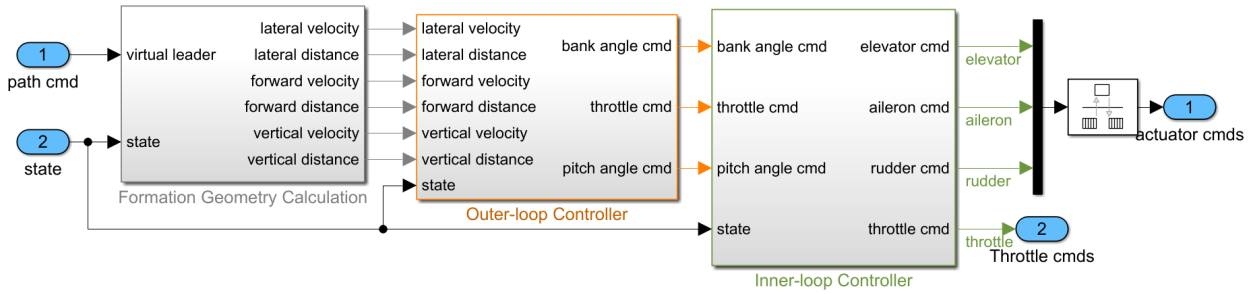


Figure 5.4 NLDI Architecture.

5.3.2.1 Outer-Loop Controller

The inversion of this controller is performed only on the forward and lateral equations. This is because the vertical tracking problem is solved by a linear expression shown in 5.25, where δ_Z and $\delta_{\dot{Z}}$ are the vertical distance and velocity errors, respectively, and K_Z with $K_{\dot{Z}}$ are their respective gains.

$$\theta_d = K_{\dot{Z}}\dot{\delta}_Z + K_z\delta_z \tag{5.25}$$

Assuming that the wingman has a coordinate turn condition, Equation 5.26 must be true. Also, assuming steady wings-level flight for the virtual trajectory, Equation 5.27 must also be true.

$$\Omega_w = \dot{\chi} \cong \frac{g}{V}tan(\phi) \tag{5.26}$$

$$\dot{\Omega_L} = 0 \tag{5.27}$$

Now, by taking the derivative of the lateral and forward parts of Equation 5.22, Equation 5.28 is obtained.

$$\begin{bmatrix} \ddot{l} \\ \ddot{f} \end{bmatrix} = \begin{bmatrix} sin(\chi_w - \chi_V) \\ -cos(\chi_w - \chi_V) \end{bmatrix}_{w_{xy}} + \begin{bmatrix} cos(\chi_w - \chi_V) \\ sin(\chi_w - \chi_V) \end{bmatrix} V_{w_{xy}}(\Omega_w - \Omega_V) + \begin{bmatrix} f \\ -l \end{bmatrix} \dot{\Omega}_V + \begin{bmatrix} \dot{f} \\ -\dot{l} \end{bmatrix} \Omega_V \quad (5.28)$$

By using an expression related to the forward translation acceleration shown in Equation 5.29 where C_D is the drag coefficient, C_Y is the side-force coefficient, g is the acceleration due to gravity, T is the Thrust and γ is the flight path angle, an expression 5.30 for the term $\dot{V}_{w_{xy}}$ can be obtained assuming a linear variation of the thrust with the throttle [53].

$$\dot{V} = \left(\frac{cos(\alpha)cos(\beta)}{m} \right) T - \left(\frac{qS(C_D cos(\bar{\beta}) - C_Y sin(\beta)}{m} + gsin(\gamma) \right) \equiv \omega_1 T - \omega_2 \quad (5.29)$$

$$\dot{V}_{w_{xy}} = \dot{V} cos(\gamma) = \frac{V_{w_{xy}}}{V} \omega_1 (T_0 + K_T \delta_T) - \frac{V_{w_{xy}}}{V} \omega_2 \quad (5.30)$$

In Equation 5.30, the constants T_0 and K_T are inherited by the propulsion model of the aircraft. Further substitutions allow the inversion of the equation, and consequently the needed output commands for the bank angle 5.31 and desired thrust 5.32 can be obtained as follows.

$$\begin{aligned} \phi_d = arctan \Bigg(& \frac{1}{gcos(\gamma)} \left[\ddot{l}_d cos(\chi_w - \chi_V) + \ddot{f}_d sin(\chi_w - \chi_V) \right] \\ & + \frac{V}{g} \Omega_V + \left[\dot{l} sin(\chi_w - \chi_V) - \dot{f} cos(\chi_w - \chi_V) \frac{\Omega_V}{gcos(\gamma)} \right] \Bigg) \end{aligned} \quad (5.31)$$

$$\begin{aligned} \delta_T = & \frac{m}{K_T cos(\gamma)} \left[\ddot{l}_d sin(\chi_w - \chi_V) - \ddot{f} cos(\chi_w - \chi_V) \right] \\ & + \frac{1}{K_T} \left[\frac{1}{2} \rho V^2 S(C_{D_0} + C_{D_\alpha} \alpha_0) + mgsin(\gamma) - T_0 \right] \\ & - \frac{m}{K_T cos(\gamma)} \Omega_V \left[\dot{l} cos(\chi_w - \chi_V) + \dot{f} cos(\chi_w - \chi_V) \right] \end{aligned} \quad (5.32)$$

Equations 5.25,5.31 and 5.32 provide the necessary outputs once the Formation Geometry Calculation is computed. These outputs will then be the inputs for the inner-loop controller as described below.

5.3.2.2 Inner-Loop Controller

The inner-loop controller is characterized by being a two-time scale inversion system consisting of a "slow mode" and a "fast mode" system. The slow mode system takes the error between the desired Euler angles $[\phi_d, \theta_d, \psi_d]'$ and the actual Euler angles $[\phi, \theta, \psi]'$ from the state of the aircraft, and outputs a vector of desired angular rates $[p_d, q_d, r_d]'$. Moreover, the fast mode system takes the error between the desired angular rates and the actual angular rates $[p, q, r]'$ from the plant, and then applies an inversion that results in commands for the aileron δ_a, elevator δ_e, and rudder δ_r. Figure 5.5 [53],[49] shows the inner-loop inversion system, which has also been widely use in more studies that have been conducted at the Advanced Dynamics and Control Lab from ERAU.

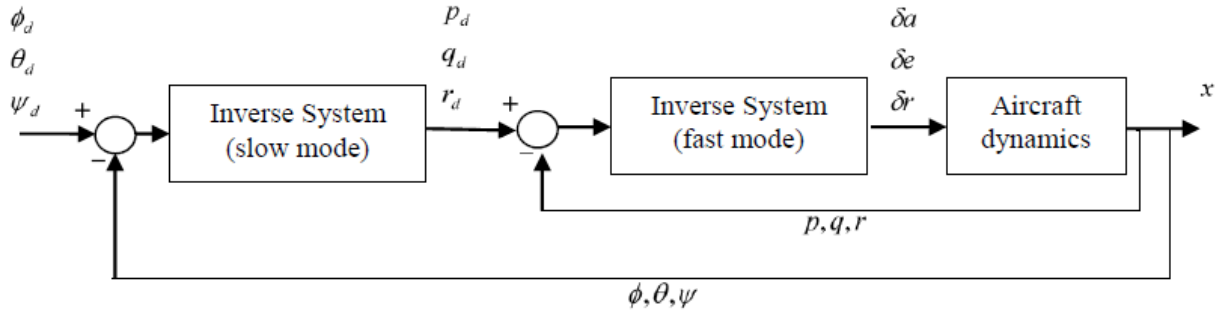


Figure 5.5 Slow and Fast modes for NLDI.

Slow Mode System

By considering the Kinematic EOMs in 5.10, the vector of angular rates $[p, q, r]'$ can be replaced by the vector of desired angular rates, and the vector of Euler rates $[\dot{\phi}, \dot{\theta}, \dot{\psi}]'$ can be replaced with a vector of pseudo controllers $[U_\phi, U_\theta, U_\psi]'$. Solving for the desired angular rates, Equation 5.33 is obtained.

$$\begin{bmatrix} p_d \\ q_d \\ r_d \end{bmatrix} = \begin{bmatrix} 1 & sin(\phi)tan(\theta) & cos(\phi)tan(\theta) \\ 0 & cos(\phi) & -sin(\phi) \\ 0 & sin(\phi)sec(\theta)cos(\phi)sec(\theta) & \end{bmatrix}^{-1} \begin{bmatrix} U_\phi \\ U_\theta \\ U_\psi \end{bmatrix} \tag{5.33}$$

The pseudo controllers are defined in Equation 5.34 as proportional controllers, where the constant gains K_ϕ, K_θ, and K_ψ are tunable parameters.

$$\begin{bmatrix} U_\phi \\ U_\theta \\ U_\psi \end{bmatrix} = \begin{bmatrix} K_\phi(\phi_d - \phi) \\ K_\theta(\theta_d - \theta) \\ K_\psi(\psi_d - \psi) \end{bmatrix} \tag{5.34}$$

Notice that the yaw angle ψ does not appear in the outer-loop of Figure 5.5, for which the third term of Equation 5.34 cannot be calculated . However, as mentioned before, a coordinate turn condition can be assumed, meaning that $\dot{\psi} = \frac{g}{V}\tan(\phi)$. Based on this assumption, the this third term can be easily calculated as follows.

$$U_\psi = \dot{\psi} = \frac{g}{V}\tan(\phi)_d \tag{5.35}$$

Now, the next step pertains to the fast mode system.

Fast Mode System

Recall the set of the aerodynamic moment equations in 5.8. Similar as for the slow mode system, the vector of moments $[L, M, N]'$ is replaced by a vector of desired moments $[L_d, M_d, N_d]'$. The rate of change of the angular rates $[\dot{p}, \dot{q}, \dot{r}]'$ is replaced by the vector of pseudo controllers $[U_p, U_q, U_r]'$ shown in Equation 5.37, where the constant gains K_p,K_q, and K_r are tunable parameters. Notice that these pseudo controllers follow the format of a proportional control, similar as before.

By rearranging the terms of the moment EOMs, Equation 5.36 is obtained.

$$\begin{bmatrix} L_d \\ M_d \\ N_d \end{bmatrix} = \begin{bmatrix} -I_{xz}pq + (I_{zz} - I_{yy})qr \\ I_{xz}(p^2 - r^2) + (I_{xx} - I_{zz})pr \\ I_{xz}qr + (I_{yy} - I_{xx}pq) \end{bmatrix} + \begin{bmatrix} I_{xx} & 0 & -I_{xz} \\ 0 & I_{yy} & 0 \\ -I_{xz} & 0 & I_{zz} \end{bmatrix} \begin{bmatrix} U_p \\ U_q \\ U_r \end{bmatrix} \tag{5.36}$$

$$\begin{bmatrix} U_p \\ U_q \\ U_r \end{bmatrix} = \begin{bmatrix} K_p(p_d - p) \\ K_q(q_d - q) \\ K_r(r_d - r) \end{bmatrix} \tag{5.37}$$

In order to obtain relationship for the control surfaces deflections, recall that the aerodynamic rolling, pitching, and yawing moment coefficients can be approximately by Taylor series functions as depicted by Equation 5.38.

$$\begin{aligned} C_l =& C_{l_0} + C_{l_\beta}\beta + \frac{b}{2V}(C_{l_p}p + C_{l_r}r) + C_{l_{\delta_a}}\delta_a + C_{l_{\delta_r}}\delta r \\ C_m =& C_{m_0} + C_{m_\alpha}\alpha + \frac{\bar{c}}{2V}C_{m_q}q + C_{m_{\delta_e}}\delta_e \\ C_n =& C_{n_0} + C_{n_\beta}\beta + \frac{b}{2V}(C_{n_p}p + C_{n_r}r) + C_{n_{\delta_a}}\delta_a + C_{n_{\delta_r}}\delta r \end{aligned} \tag{5.38}$$

Recall also that the aerodynamic moments can also be expressed as a function of their respective coefficients as in Equation 5.39.

$$\begin{bmatrix} L_d \\ M_d \\ N_d \end{bmatrix} = \bar{q}S \begin{bmatrix} bC_l \\ \bar{c}C_m \\ bC_n \end{bmatrix} \tag{5.39}$$

For instance, by replacing Equation 5.38 in Equation 5.39, expressions for the control surface deflections in terms of the known moments and geometrical variables can be obtained. Therefore, the equation for the elevator deflection is as follows.

$$\delta_e = \frac{\frac{M_d}{\bar{q}S\bar{c}} - C_{m_0} - C_{m_\alpha}\alpha - \frac{\bar{c}}{2V}C_{m_q}q}{C_{m_{\delta_e}}} \tag{5.40}$$

Notice, however, that due to the rolling moment being coupled with the yawing moment, a system of two equations must be solved in order to obtain the aileron and rudder deflections. For simplification, let:

$$\begin{aligned} b_1 =& C_{l_{\delta_a}} \delta_a + C_{l_{\delta_r}} \delta r \\ b_2 =& C_{n_{\delta_a}} \delta_a + C_{n_{\delta_r}} \delta r \end{aligned} \tag{5.41}$$

By solving the mentioned two-system of equations, the final expression for the aileron and rudder deflections are presented in Equation 5.42.

$$\begin{aligned} \delta_a =& \frac{C_{l_{\delta_r}} b_2 - C_{n_{\delta_r}} b_1}{C_{l_{\delta_r}} C_{n_{\delta_a}} - C_{n_{\delta_r}} C_{n_{\delta_a}}} \\ \delta_r =& \frac{C_{n_{\delta_a}} b_1 - C_{l_{\delta_a}} b_2}{C_{l_{\delta_r}} C_{n_{\delta_a}} - C_{n_{\delta_r}} C_{n_{\delta_a}}} \end{aligned} \tag{5.42}$$

With these equations, the fast mode system is completed, leading to the completion of the inner-loop controller. By taking the inputs of the outer-loop controller, which at the same time takes the inputs of the Formation Geometry Calculation, now the controller is able to command the control surfaces of the Rascal 110 as expected, based on a trajectory commanded by the RRT* planner.

5.4 Architectures for Simulation Environment

Interactions between Matlab/Simulink environments are used to simulate the results of this study. Figure 5.6 shows a scheme of how the HMS is interconnected with RRT* path planning algorithm and the generation of safe trajectories. The goal is to provide the system an efficient and safe trajectory that can be automatically generated when the AIS paradigm detects and identifies a low or high magnitude failure.

For this thesis, the RRT* path planner is build in Matlab while the dynamics of the UAV, the replanning logic, and the HMS are build in Simulink. Figure 5.7 provides the general architecture of the Rascal Simulink model, which also includes an animation block and an interface to FlightGear Version 2020.3.9 for visualization purposes.

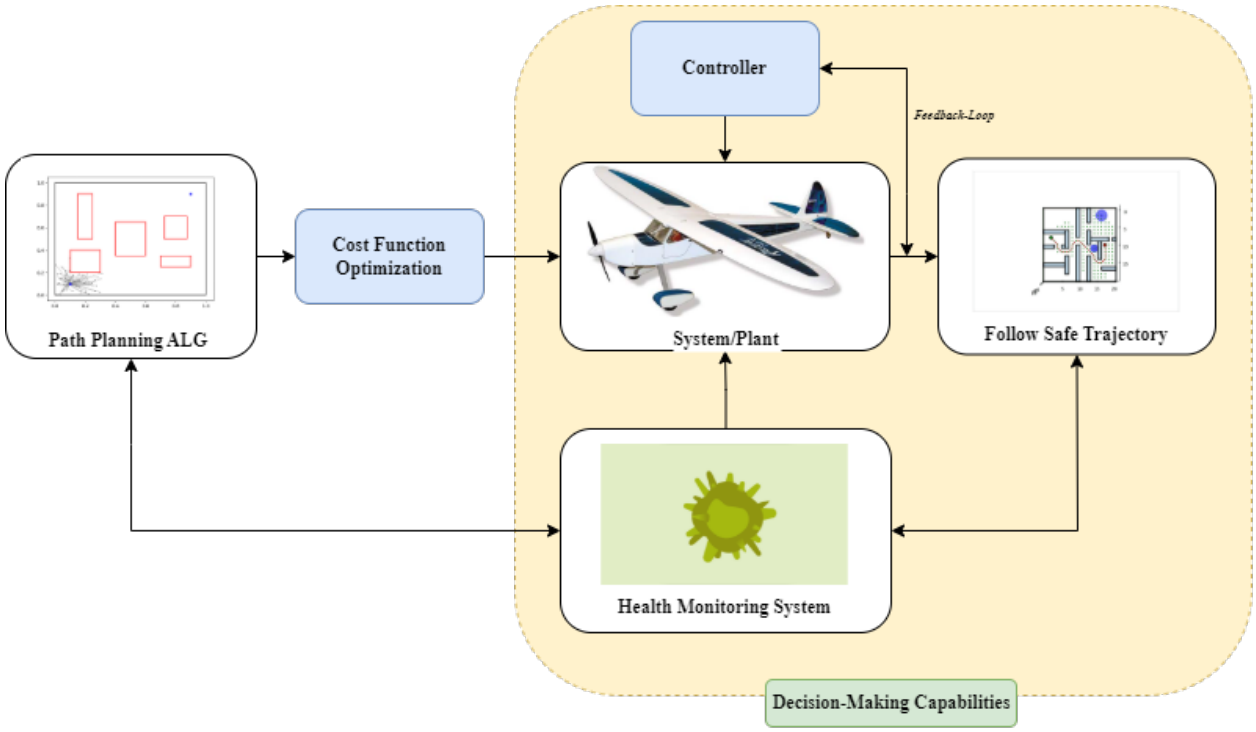


Figure 5.6 General Simulation Architecture.

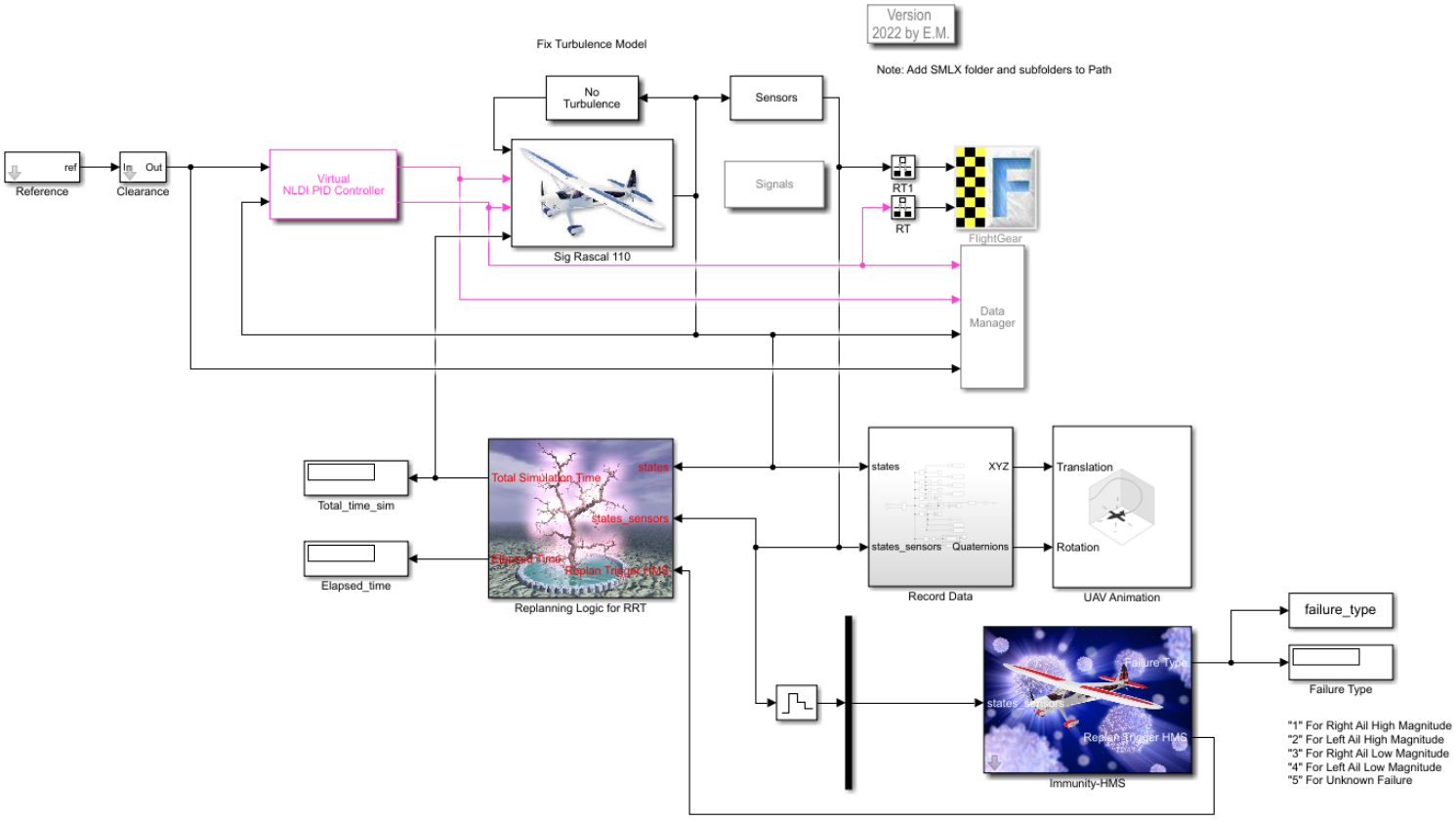


Figure 5.7 Simulink Model with FlightGear.

The algorithm flow shown in Figure 5.8 presents the proposed architecture logic for the on-board decision-making capabilities of the fixed-wing. The algorithm starts by uploading the 3D Occupancy map into the OBC, and then the user selects n poses to be visited during the initial mission planning phase. After this, the RRT* Multi-goal planner plans a path towards each pose until the final pose is reached. The UAV starts its flight, and through the HMS, a constant check for failures is performed. If a failure is detected, the AIS paradigm enters into action by identifying the type of failure and applying metrics to evaluate the flight envelope. This information is passed towards the RRT* Multi-goal planner containing the same initial poses, and the new safe trajectory is generated such that the fixed-wing is able to accomplish the initial mission with an active failure.

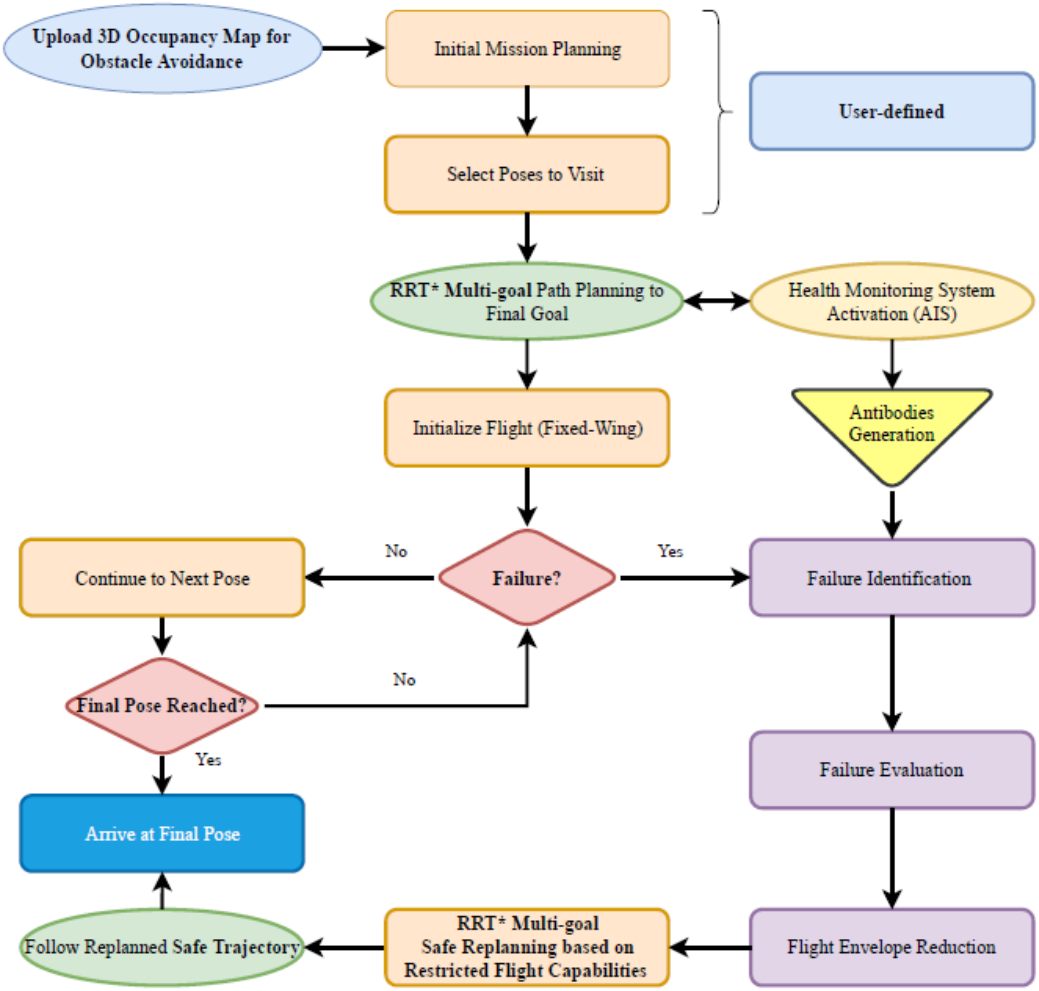


Figure 5.8 On-board Decision-making Capabilities Architecture.

Finally, Figure 5.9 shows an example of the visual simulation that is product of the planning and replanning missions mentioned above. For best computational performance, the PC used to carry all the simulations and algorithms of this study had an Intel(R) Core(TM) i7-10510U CPU @ 1.80GHz 2.30 GHz processor with 16.0 GB RAM memory, and Windows 10 Pro.

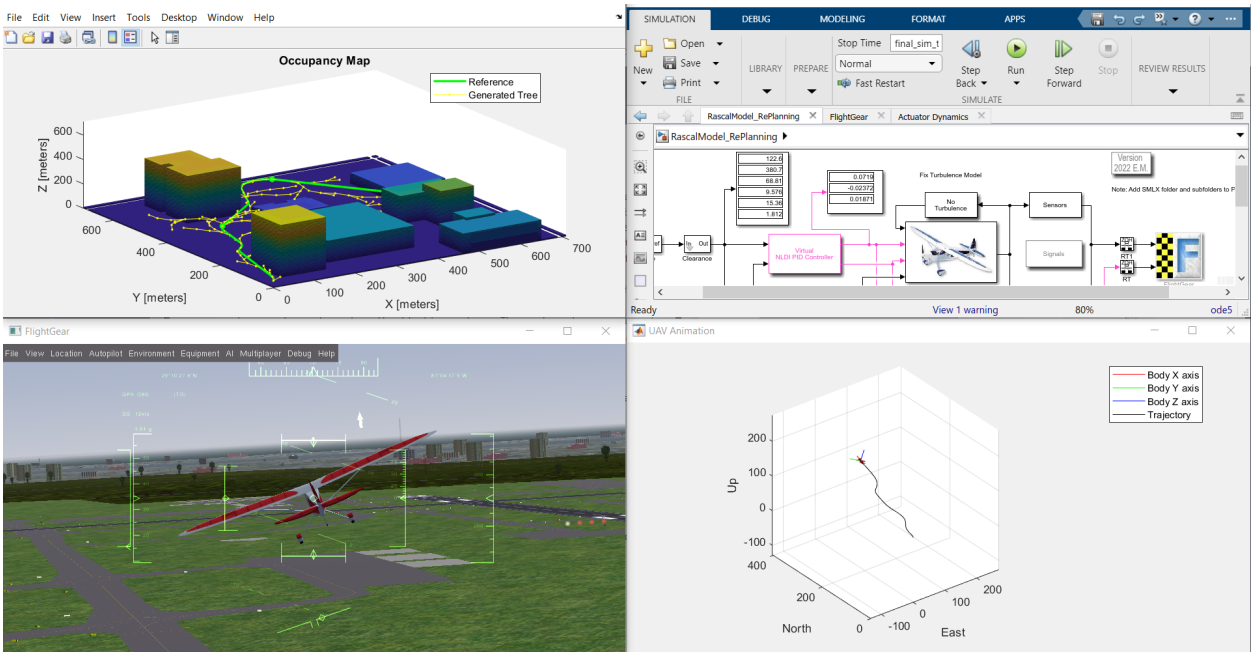


Figure 5.9 Example of Trajectory Tracking: Generated RRT* Trajectory (top left), Rascal 110 Simulink Model (top right), FlightGear Simulator (bottom left), and UAV Animation (bottom right).

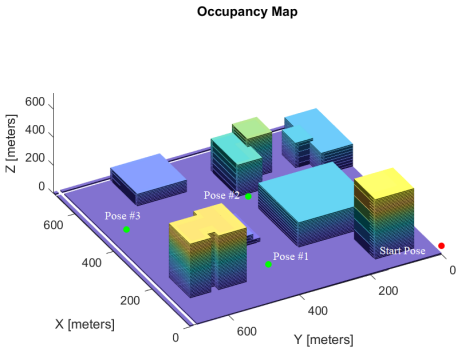
6 Numerical Simulations & Performance Analysis

6.1 Nominal Trajectory Generation

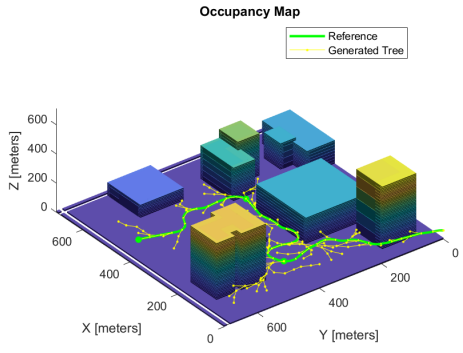
A simple mission with nominal waypoints at different altitudes has been created on the 3D map described before to test the proposed architecture. Table 6.1 shows the coordinates of the poses to visit, including the desired headings. Furthermore, Figure 6.1 shows the generated trajectory before and after the application of the Dubins Paths for smoothing.

Table 6.1 Nominal trajectory coordinates.

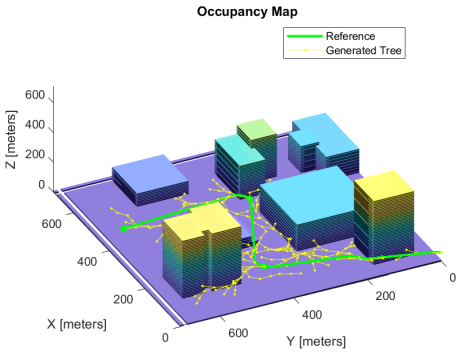
	Cartesian Coordinates			
Pose	**X (m)**	**Y(m)**	**Z (m)**	**Heading (deg)**
Start	0	0	60	45
#1	126	420	65	45
#2	420	317	80	0
#3	440	650	60	90



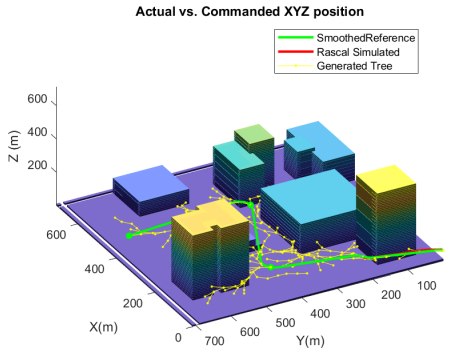
(a) Initial Defined Poses.



(b) Original Path without smoothing.



(c) Path with Dubins smoothing.



(d) Rascal Actual Trajectory Following.

Figure 6.1 Nominal Trajectory Generation.

Additionally, Figure 6.1d plots the simulated trajectory followed by the Rascal 110. It can be seen that the NLDI controller is robustly enough such that the actual simulated trajectory is almost indifferent to the reference trajectory at nominal conditions. Using the data recorded from this flight, the nominal selves are generated and the described failures are introduced as a part of the supervised learning process of the HMS.

6.2 Replanning Missions due to Aileron Failure

Each of the failures described by Table 4.1 restricts the flight envelop in terms of the maximum roll angle that can be achieved on each direction. After applying the equations shown in subsection 4.2.1, new limits on the roll angle are obtained according to each specific failure type. Table 6.2 shows these new limits, where the direction of the limitation is also specified.

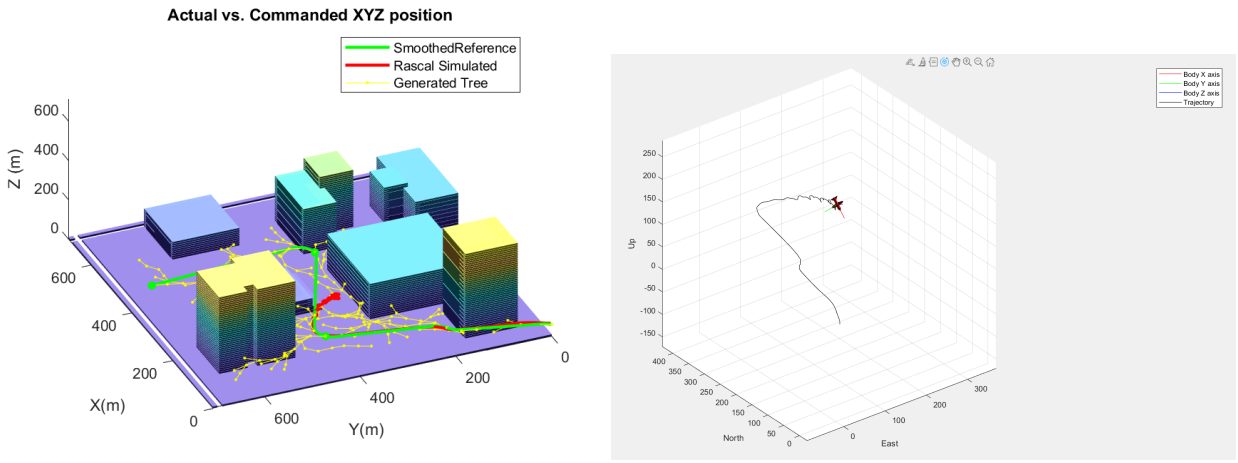
Table 6.2 Roll Angle Restrictions.

Aileron Failure Case	**Nominal Max. Roll (deg)**	**Restricted Roll Angle (deg)**
$\delta_{ail_{R_{stuck}}} = 15°$	50	14.2857 (max.right)
$\delta_{ail_{R_{stuck}}} = -10°$		27.0270 (max.left)
$\delta_{ail_{L_{stuck}}} = 17°$		11.3636 (max.left)
$\delta_{ail_{L_{stuck}}} = -8°$		22.8571 (max.right)

This information is given manually to the planner at the time of failure when the supervised training process is done. However, when the HMS is activated with the AIS paradigm, the information is passed automatically to the planner when the detection and identification algorithms are activated such that a closed-loop between the planner and the HMS is achieved.

6.2.1 Right Aileron Failure (High Magnitude)

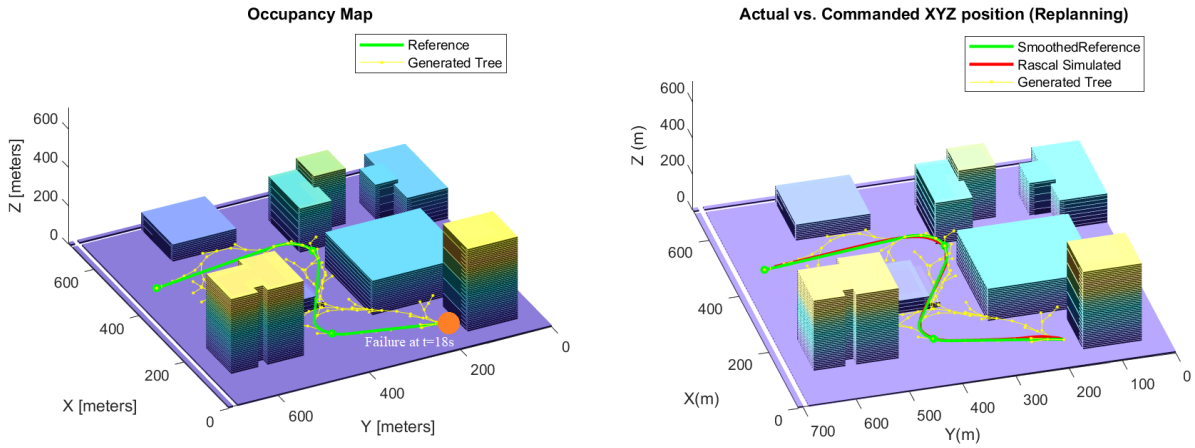
Consider the first case of failure described by Table 4.1, where the right aileron is stuck at $\delta_{ail_{R_{stuck}}} = 15°$ at $t = 18s$. For this case, the controller is able to counteract the failure but just to a certain point. Figure 6.2 shows the simulation where the vehicle is not able to follow the nominal trajectory once it reaches the first turn to the right due to the reduced maximum achievable roll angle in this direction. Since the right aileron is inducing the aircraft roll to the left constantly, a new trajectory must be generated for saving the aircraft from collision.



(a) Rascal Simulated Trajectory. *(b)* Rascal 110 Crash Animation.

Figure 6.2 Rascal 110 Crash for $\delta_{ail_{R_{stuck}}} = 15°$.

Figure 6.3a shows the new replanning mission generated by the decision-making architecture of the path planner. Notice that the trajectory has been modified by inserting higher turning radii in the turns. Figure 6.3b demonstrates that the aircraft is now able to follow the new path with approximately the same flight time. The NLDI controller does its job and the original waypoints of the mission are visited, which guarantees the safety not only for the vehicle, but for the completion of the mission while preventing collision or damage risks.



(a) Generated Safe Trajectory. *(b)* Rascal 110 Trajectory Simulation.

Figure 6.3 Rascal 110 Replanning Mission for $\delta_{ail_{R_{stuck}}} = 15°$.

Figure 6.4 shows a clear difference between the replanned and the original smoothed trajectory from the top view of the map. Though both trajectories seem to be similar, they cause completely different responses from the aircraft once the AC is detected.

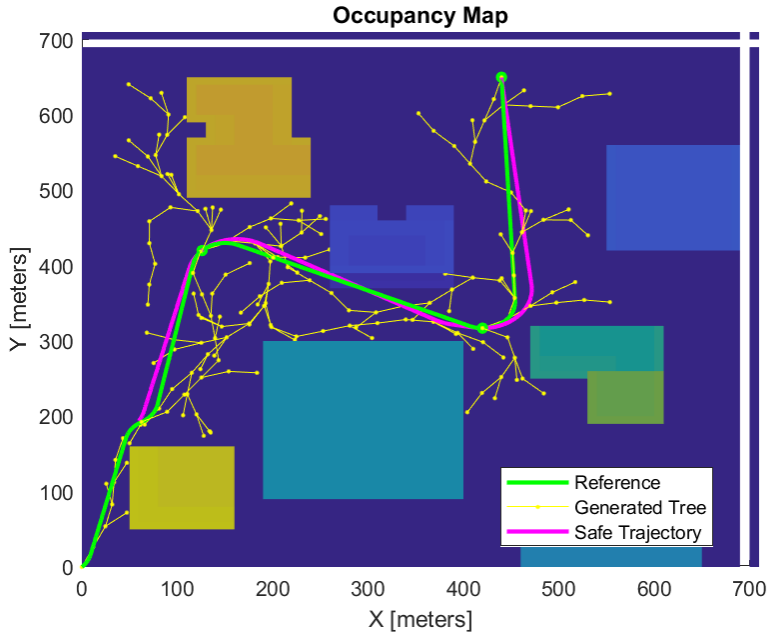
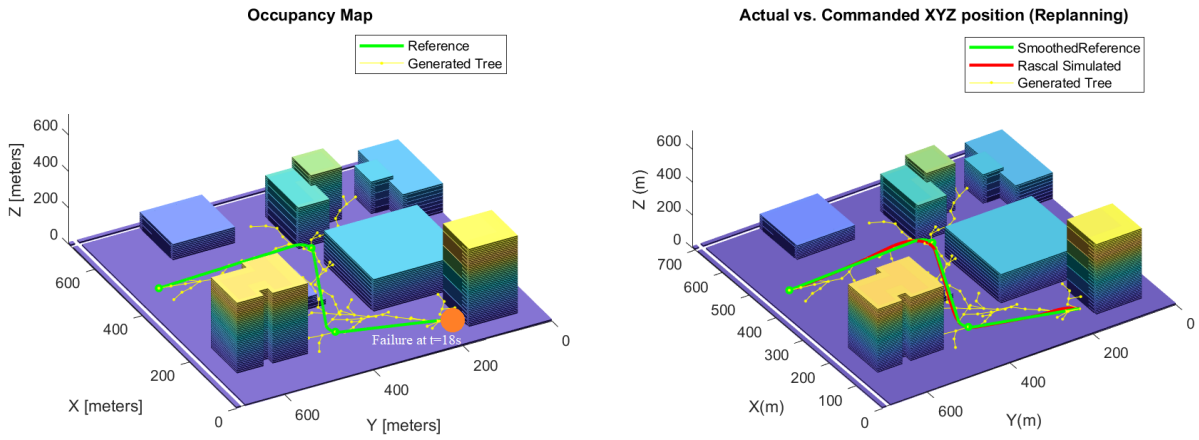


Figure 6.4 Smoothed Nominal Trajectory vs. Replanned Safe Trajectory for Right Aileron Failure (High Magnitude).

6.2.2 Right Aileron Failure (Low Magnitude)

Within the same line of thought, now a lower magnitude failure deflection $\delta_{ail_{R_{stuck}}} = -10°$ is inserted at the same flight time as before. This failure will restrict the capability of the aircraft to roll to the left by pushing the aircraft to roll right. Figure 6.5 shows the replanning mission for this case. An interesting thing to notice is that the NLDI controller is able to compensate for this failure, and the aircraft is able to complete the mission. However, the replanning algorithm still generates a new different path. This path is in accordance with the flight envelope restriction, and as will be shown in Section 6.3, it improves the overall performance of the UAV at post-failure conditions in comparison with the case when the replanning is deactivated.



(a) Generated Safe Trajectory. *(b)* Rascal 110 Trajectory Simulation.

Figure 6.5 Rascal 110 Replanning Mission for $\delta_{ail_{R_{stuck}}} = -10°$.

Figure 6.6 shows the comparison between the nominal and the safe trajectory, respectively. Notice that the turning radii are constrained differently and the generated tree presents slightly variations in comparison with the tree from Figure 6.4.

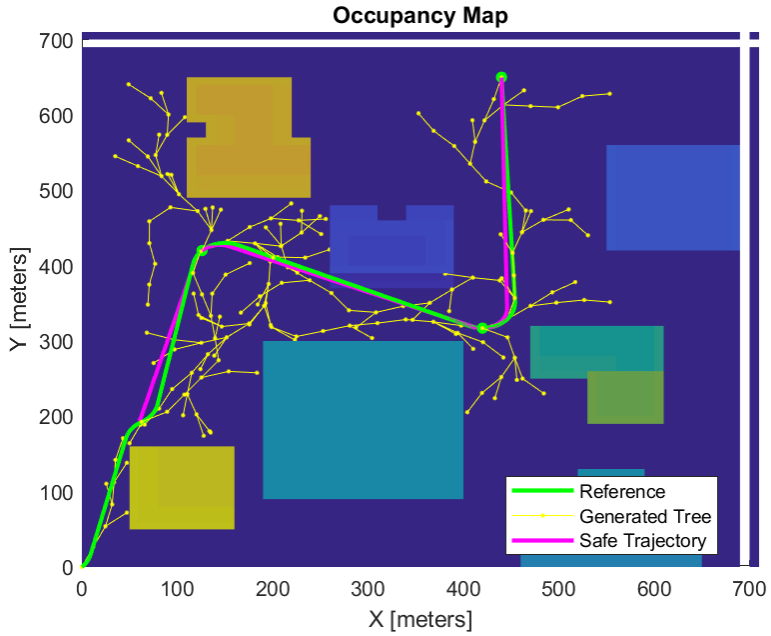
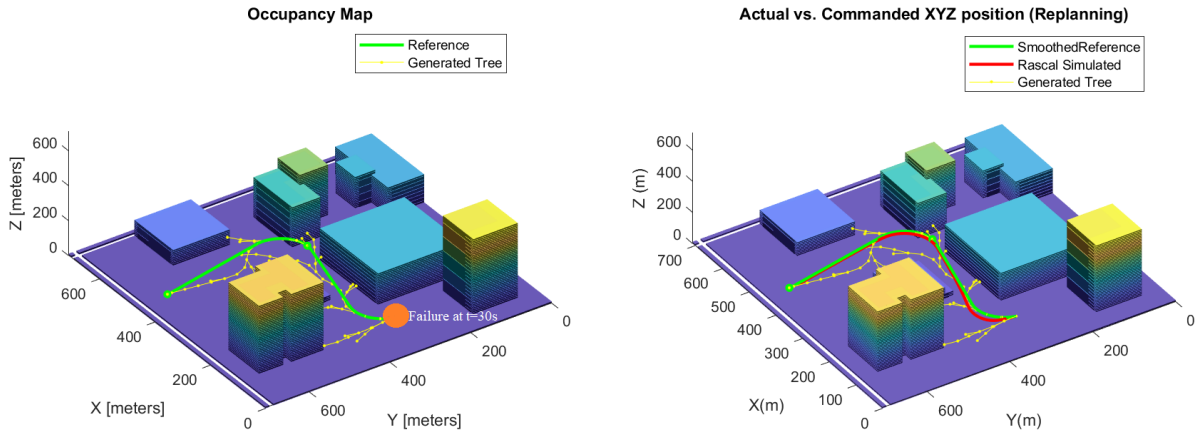


Figure 6.6 Smoothed Nominal Trajectory vs. Replanned Safe Trajectory for Right Aileron Failure (Low Magnitude).

6.2.3 Left Aileron Failure (High Magnitude)

Now, for the third case, consider the high magnitude failure of the left aileron stuck at $\delta_{ail_{L_{stuck}}} = 17°$ at $t = 30s$. Figure 6.7 shows the generated safe trajectory. Similar to the previous case, the aircraft is able to follow the trajectory due to the robustness of the controller. However, notice that the replanned mission increments again the turning radii due to the aircraft tendency to roll to the right at post-failure condition. Notice that the planner modifies also the straight path to be followed after the remaining turn to the left.



(a) Generated Safe Trajectory. *(b)* Rascal 110 Trajectory Simulation.

Figure 6.7 Rascal 110 Replanning Mission for $\delta_{ail_{L_{stuck}}} = 17°$.

An interesting highlight for this case is that the new safe trajectory does not completely pass through pose #1, as can be seen in the comparison of the trajectories presented on Figure 6.8. Due to the high magnitude failure, the decision-making algorithm of the planner skips this waypoint and continuous to replan the mission for guaranteeing the safety of the vehicle. The left aileron lock downwards pushes the system to try to roll right, for which the needed left turns will require more time to be completed, thereby justifying increase in left turning radius.

Despite this, the question of whether the replanning mission is needed or not still remains as the mission without replanning is still being completed. Again, the answer relies on the overall performance of the aircraft that will be further presented in the next section.

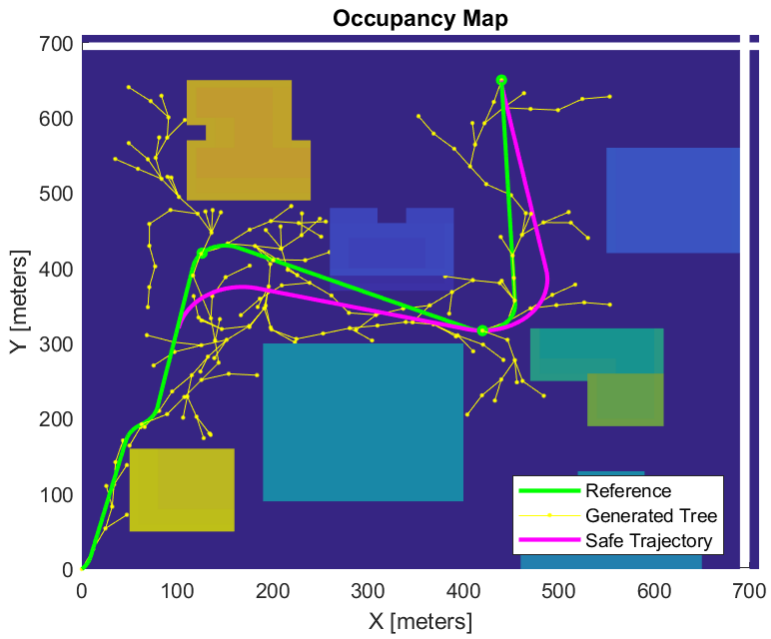
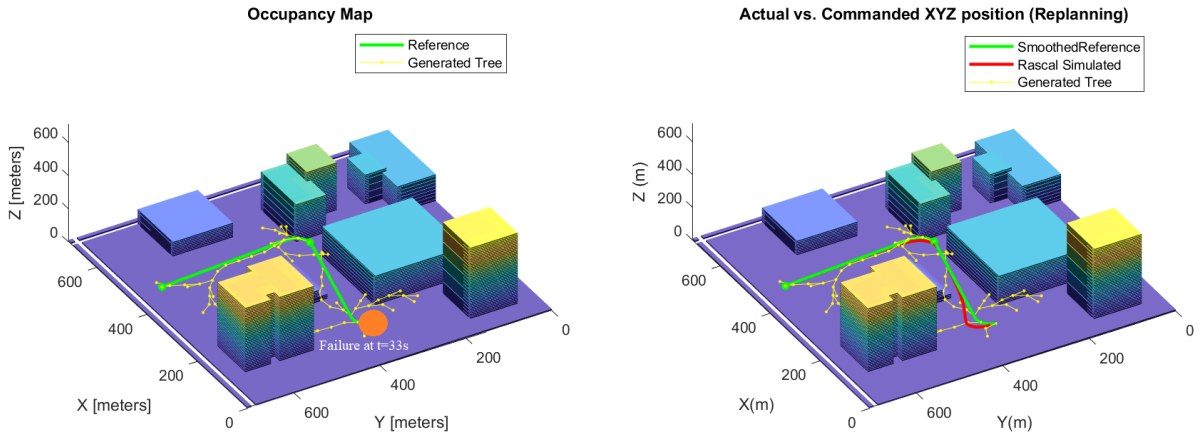


Figure 6.8 Smoothed Nominal Trajectory vs. Replanned Safe Trajectory for Left Aileron Failure (High Magnitude).

6.2.4 Left Aileron Failure (Low Magnitude)

The left aileron has been stuck at $\delta_{ail_{L_{stuck}}} = -8°$ at $t = 33s$. Figure 6.9 shows the result of the replanning mission. Once again, the aircraft does not crash with this AC, but the replanning mission leads to the creation of a new trajectory that avoids visiting pose #1.



(a) Generated Safe Trajectory.

(b) Rascal 110 Trajectory Simulation.

Figure 6.9 Rascal 110 Replanning Mission for $\delta_{ail_{L_{stuck}}} = -8°$.

Figure 6.10 shows how the nominal trajectory is modified. The aircraft is able to make the designated original turn to the left according to the planner, but still pose #1 is being avoided to ensure a safe trajectory.

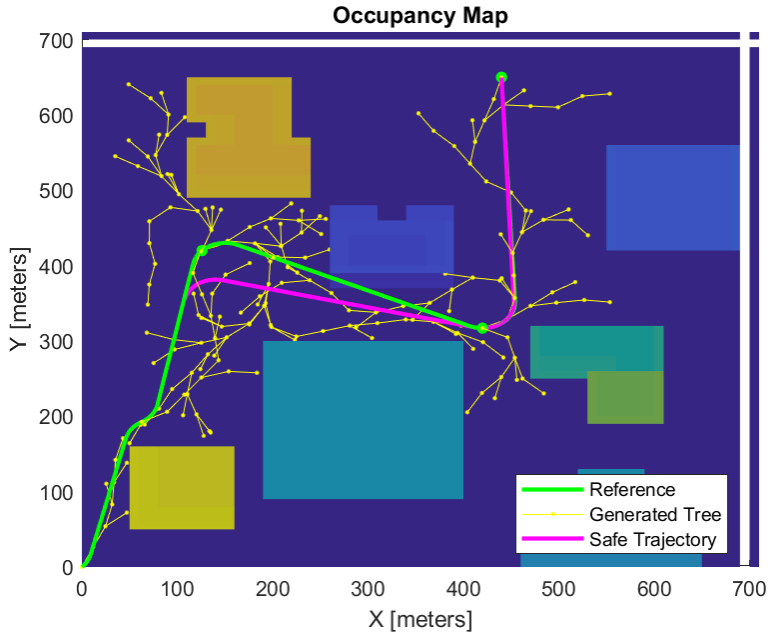


Figure 6.10 Smoothed Nominal Trajectory vs. Replanned Safe Trajectory for Left Aileron Failure (Low Magnitude).

6.3 Performance Analysis of Generated Safe Trajectories

6.3.1 Path Planner Costs

The evaluation of the performance of the path planner is addressed by the total length of the generated safe trajectories as well as the iterations and number of nodes used to reach the goal position. Table 6.3 shows these calculations for the failure cases and the nominal path.

Table 6.3 Path Planner Costs and Iterations

Failure Case #	Nominal Path Cost	Nominal # of Total Iterations	Nominal # of Total Nodes	Replanned Path Cost	Replanned # of Total Iterations	Replanned # of Total Nodes
1	1283.1	468	184	968.92	157	117
2				1024.5	171	111
3				1155.5	188	138
4				1126.5	218	139

It can be seen that the lengths of the path for the failures that are detected at approximately the same time are close to each other. Additionally, all the replanned paths have lesser path costs, iterations, and number of generated nodes than the nominal path. This is of essential importance since the flight envelop restriction would tend to force the path planner to create more complicated trajectories that might end up having higher costs. In fact, this is one more reason that justifies the usage of the RRT* planner instead of other path planners.

6.3.2 Performance Metrics for Evaluating Trajectory Feasibility

When the failure is counteracted by the NLDI controller (as happened for cases 2-4), a question arises that proposes whether the replanning mission is worth to be done or not. For evaluating the value of the replanning architecture demonstrated in this study, new metrics are defined in terms of how well the vehicle tracks the reference trajectory in terms of Cartesian coordinates, Euler angles, and the energy used for the control surfaces actuation [54]. For this, let PI_{global} be a global Performance Index defined by Equation 6.1:

$$PI_{global} = 1 - [w_1\tilde{T} + w_2\tilde{E} + w_3\tilde{C}] \tag{6.1}$$

where: $\tilde{T}$ and $\tilde{E}$ are the accumulated errors of translational displacement and attitude error tracking, respectively, and $\tilde{C}$ is the accumulated energy spent by the ailerons, elevators, and rudder during the entire length of the trajectory. The coefficients w_1, w_2, and w_3 are simply weighting factors that are chosen to give different level of importance to each metric.

Equation 6.2 defines the coordinate tracking metric, where the error integrals T_x, T_y, and T_z are defined in Equation 6.3 based on the flight time T_f. The coefficients C_x, C_y, and C_z are normalization factors that are obtained from the worst case performance of all the simulations.

$$\tilde{T} = \frac{1}{3}[T_x\ T_y\ T_z] \begin{bmatrix} 1/C_x \\ 1/C_y \\ 1/C_z \end{bmatrix} \tag{6.2}$$

$$T_x = \sqrt{\frac{1}{T_f}\int_0^{T_f} e_x^2\, dt} \qquad T_y = \sqrt{\frac{1}{T_f}\int_0^{T_f} e_y^2\, dt} \qquad T_z = \sqrt{\frac{1}{T_f}\int_0^{T_f} e_z^2\, dt} \tag{6.3}$$

Equation 6.4 defines the attitude error tracking based only on the pitch and roll angles, which are the most affected ones by the aileron failures. Same as before, the error integrals are defined in Equation 6.5 and the C-coefficients also come from the worst case performance.

$$\tilde{E} = \frac{1}{2}[E_\theta\ E_\phi] \begin{bmatrix} 1/C_\theta \\ 1/C_\phi \end{bmatrix} \tag{6.4}$$

$$E_\theta = \sqrt{\frac{1}{T_f}\int_0^{T_f} e_\theta^2\, dt} \qquad E_\phi = \sqrt{\frac{1}{T_f}\int_0^{T_f} e_\phi^2\, dt} \tag{6.5}$$

Finally, Equation 6.6 accounts for the energy spent on the actuation. The integrals shown in Equation 6.7, for this case, are composed by the absolute value of the rate of change of each control surface, which is then integrated over the entire flight time. The C-coefficients are then used again for normalization purposes.

$$\tilde{C} = \frac{1}{3}[C_{ail}\ C_{ele}\ C_{rud}] \begin{bmatrix} 1/C_{\delta_{ail}} \\ 1/C_{\delta_{ele}} \\ 1/C_{\delta_{rud}} \end{bmatrix} \tag{6.6}$$

$$C_{\delta_{ail}} = \sqrt{\frac{1}{T_f}\int_0^{T_f} |\dot{\delta_{ail}}(t)|\, dt} \quad C_{\delta_{ele}} = \sqrt{\frac{1}{T_f}\int_0^{T_f} |\dot{\delta_e}(t)|\, dt} \quad C_{\delta_{rud}} = \sqrt{\frac{1}{T_f}\int_0^{T_f} |\dot{\delta_{rud}}(t)|\, dt} \tag{6.7}$$

It is considered that the tracking of Euler angles and the energy of the actuation has more weight in the general performance of the new trajectories rather than the trajectory tracking per se. For instance, the first weighting factor is assigned to be $w_1 = 5\%$, while

$w_2 = 60\%$ and $w_3 = 35\%$. Table 6.4 presents the results that have been obtained by enabling and disabling the online HMS with the replanning algorithm for all the cases.

Table 6.4 Performance Metrics for Simulations.

# Simulation Case		$\tilde{\mathbf{T}}$	$\tilde{\mathbf{E}}$	$\tilde{\mathbf{C}}$	$\mathbf{PI_{global}}$
Nominal Trajectory		0.8262	0.0463	0.2920	0.8287
1	Right Aileron Stuck at 15° NO Replanning	**Crash**			
	Right Aileron Stuck at 15° with Replanning	**0.6149**	**0.0692**	**0.4100**	**0.7842**
2	Right Aileron Stuck at -10° NO Replanning	0.8105	0.1071	0.5252	0.7114
	Right Aileron Stuck at -10° with Replanning	**0.6015**	**0.0657**	**0.3699**	**0.8010**
3	Left Aileron Stuck at 17° NO Replanning	0.8146	0.0783	0.4439	0.7569
	Left Aileron Stuck at 17° with Replanning	**0.8218**	**0.0644**	**0.3308**	**0.8045**
4	Left Aileron Stuck at -8° NO Replanning	0.7999	0.0623	0.3526	0.7992
	Left Aileron Stuck at -8° with Replanning	**0.6633**	**0.0626**	**0.3333**	**0.8126**

Notice that for all the cases, the global PI improves if the replanning mission is activated. For the first case, the replanning algorithm saves the mission, and the new safe trajectory has even a high global PI close to the nominal value. For the second case, less actuation energy is spent and the Euler tracking is better, which improves significantly the global PI. The third case is similar to the second case, although the trajectory tracking has greater cost with the replanned mission. For the last case, the Euler tracking is approximately the same, but the cost of the actuation energy and trajectory tracking is less for the replanned mission, consequently increasing the global PI almost to the nominal value.

These results demonstrate that the consideration of the kinodynamic properties of the fixed-wing has a great effect in path planning missions, especially when the flight envelop is reduced. In general, less actuation energy is spent while a better tracking of the attitude is achieved, which makes the trajectory safer to be followed.

6.4 HMS-AIS Detection and False Alarms

The antibodies have been generated using 2000 clusters with a minimum size of 0.005. A maximum number of 500 detectors are inserted within a limit of 400 iterations, all with a tolerance of 0.0001 for every generated cluster.

With this, in order to evaluate the performance of the AIS, recall Equation 4.23 and Equation 4.24 which describe the Detection Rates and the False Alarms. Table 6.5 and Table 6.6 present the calculations for the right and left aileron failure cases, respectively.

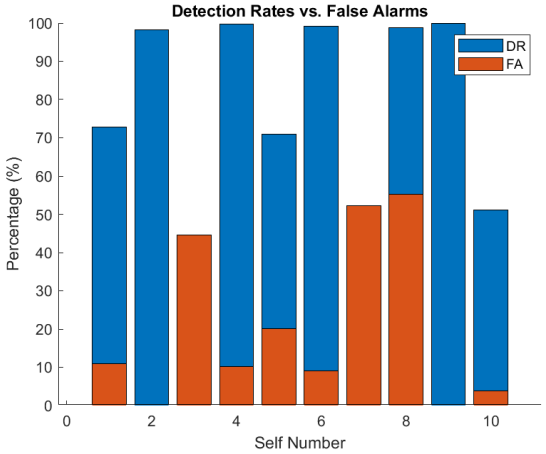
Table 6.5 DR and FA for Right Aileron Failure Cases.

		Right Aileron Stuck at 15°, t=18s		**Right Aileron Stuck at -10°, t=18s**	
#Self	**Features**	**DR (%)**	**FA (%)**	**DR (%)**	**FA (%)**
1	p,δ_{Ail_L}	72.82	11.01	77.90	4.12
2	p,δ_{Ail_R}	98.30	0.00	99.93	0.00
3	$pb/2V$,δ_{Ail_L}	33.75	44.49	82.41	11.46
4	$pb/2V$,δ_{Ail_R}	99.75	10.12	99.80	7.56
5	r,δ_{Ail_L}	70.90	20.13	81.66	4.89
6	r,δ_{Ail_R}	99.10	9.01	98.90	28.70
7	δ_{Ail_L},ϕ	46.02	52.28	94.23	27.25
8	δ_{Ail_R},ϕ	98.80	55.28	98.83	9.23
9	δ_{Rud_L},δ_{Ail_R}	99.83	0.00	99.93	0.00
10	δ_{Rud_R},δ_{Ail_L}	51.06	3.89	83.31	2.22

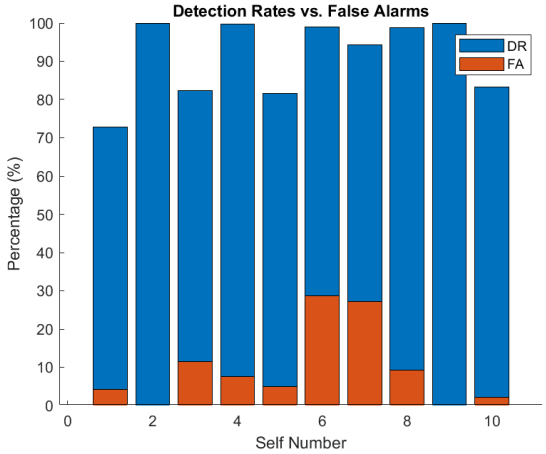
Table 6.6 DR and FA for Left Aileron Failure Cases.

		Left Aileron Stuck at 17°, t=30s		**Left Aileron Stuck at -8°, t=33s**	
#Self	**Features**	**DR (%)**	**FA (%)**	**DR (%)**	**FA (%)**
1	p,δ_{Ail_L}	89.43	0.00	41.59	0.00
2	p,δ_{Ail_R}	87.93	0.00	13.59	7.87
3	$pb/2V$,δ_{Ail_L}	72.27	9.27	27.25	1.60
4	$pb/2V$,δ_{Ail_R}	94.58	10.87	16.31	7.54
5	r,δ_{Ail_L}	63.10	0.00	37.62	6.34
6	r,δ_{Ail_R}	91.43	0.00	26.48	15.74
7	δ_{Ail_L},ϕ	93.83	0.00	91.26	0.00
8	δ_{Ail_R},ϕ	93.55	0.00	33.95	64.38
9	δ_{Rud_L},δ_{Ail_R}	93.40	0.00	17.46	8.41
10	δ_{Rud_R},δ_{Ail_L}	98.85	0.00	71.67	0.00

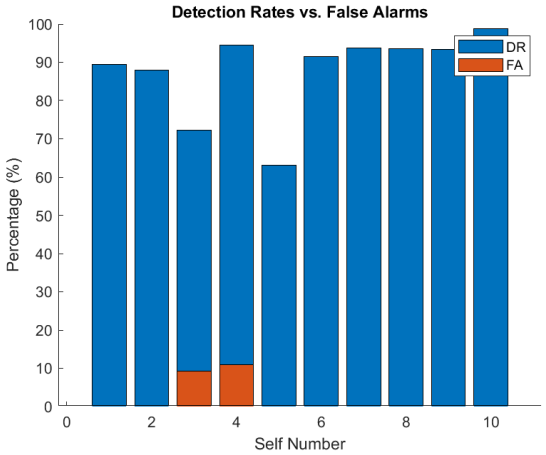
From these calculations, it can be seen that the FAs are much more lower than the DRs, which indicates an acceptable performance of the AIS. The relationship between DRs and FAs depends on the type failure inserted and the detection time. Despite the fact that in some cases the FAs are higher than the DRs for certain selves, still the detection and identification algorithms work as expected. Figure 6.11 confirms this comparison within all the cases.



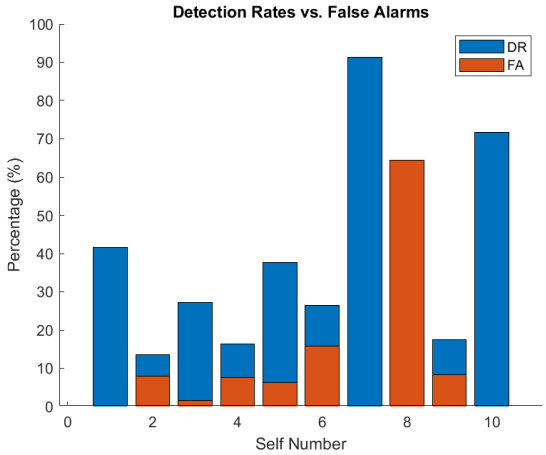
(a) DRs and FAs for Case 1.



(b) DRs and FAs for Case 2.



(c) DRs and FAs for Case 3.



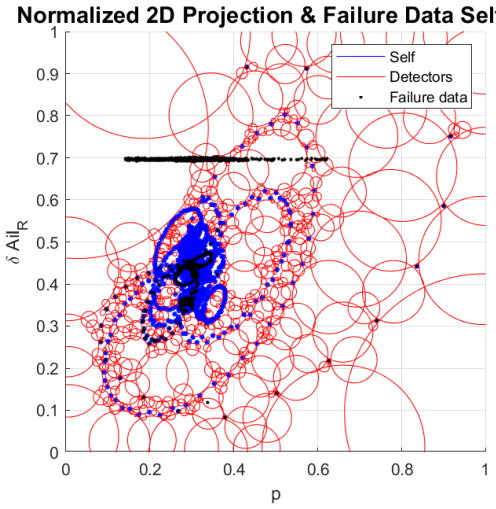
(d) DRs and FAs for Case 4.

Figure 6.11 DRs and FAs for all Simulation Cases.

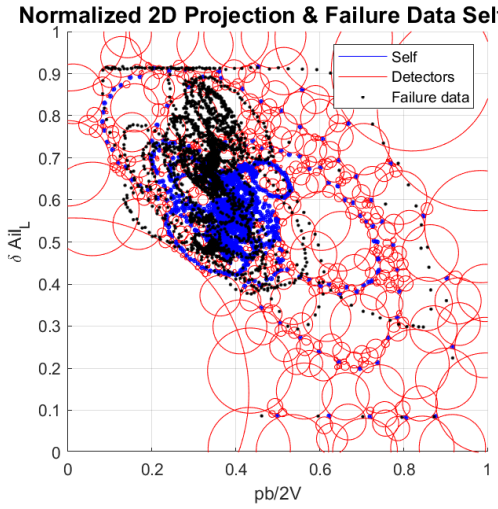
The worst performance can be seen to be case 4, which is the low magnitude failure for the left aileron. This could be improved by generating more self clusters at the cost of requiring more computational effort.

6.4.1 Generated Selves for Detection and Identification

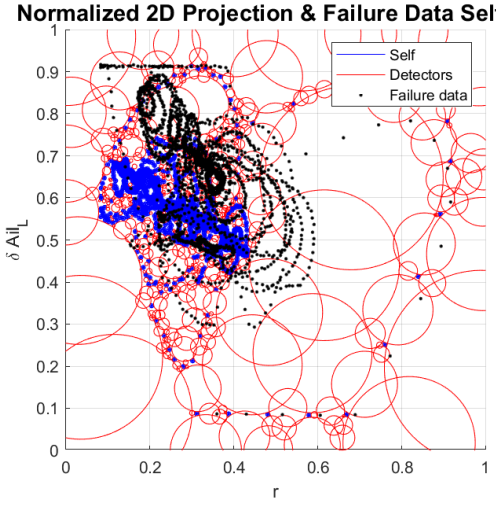
After the offline training process has been finalized, the projections can be visualized. Figure 6.12 shows some selves out of the 10 selves, and the failure data plotted for case 1. The failure of the right aileron is evident especially in Figure 6.12a and Figure 6.12d where constant "lines" of data points can be seen at normalized high magnitudes. Notice, however, that Figure 6.12b and Figure 6.12c present the same constant lines, but they are shorter and not so well defined because this is the left aileron trying to compensate for the failure.



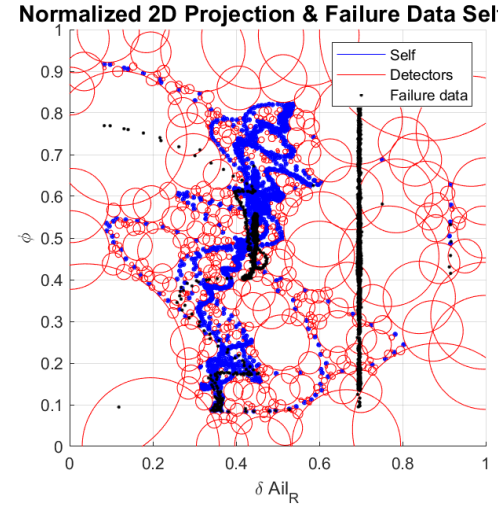
(a) Self 2 for Case 1.



(b) Self 3 for Case 1.



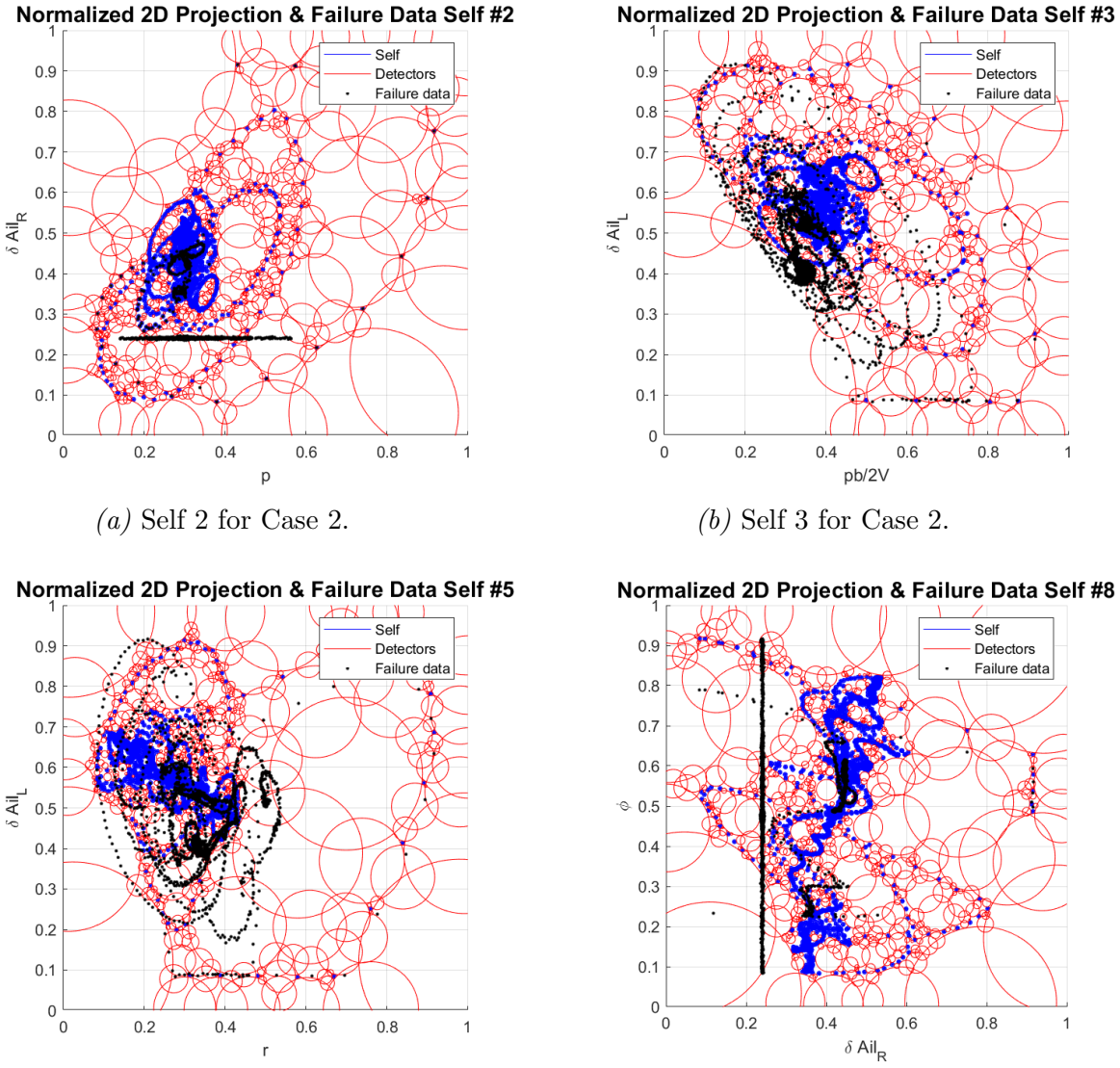
(c) Self 5 for Case 1.



(d) Self 8 for Case 1.

Figure 6.12 Example of Selves for Failure Case 1: Right Aileron Stuck (High Magnitude).

Figure 6.13 below shows the same number of selves but for case 2. Notice that the constant lines in Figure 6.13a and Figure 6.13d have changed positions since the aileron is stuck in the upright position. Also, Self 3 in Figure 6.13b and Self 5 in Figure 6.13c show that the left aileron is not doing too much effort to counteract the failure in comparison with case 1. Notice also that the data points in these two selves have shifted position since the left aileron is being deflected in the opposite direction than before.



(a) Self 2 for Case 2.

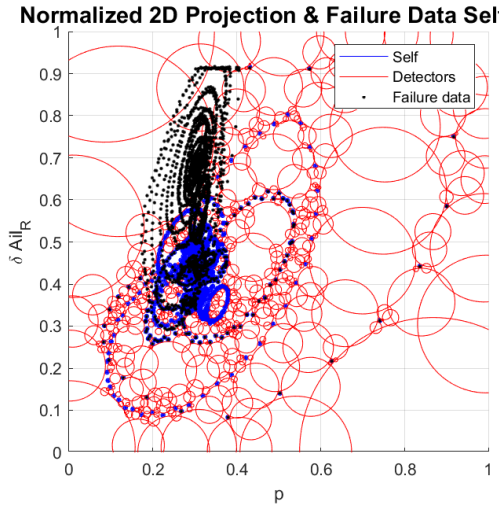
(b) Self 3 for Case 2.

(c) Self 5 for Case 2.

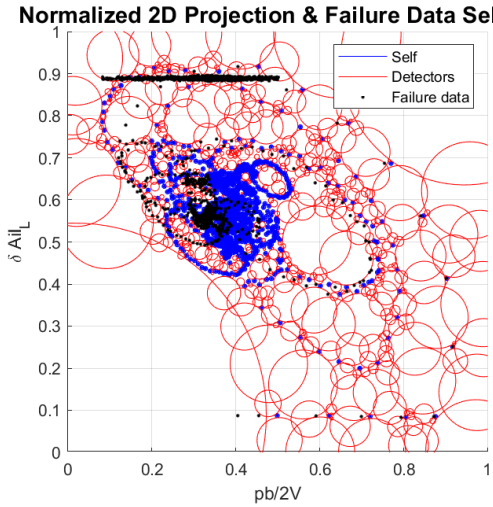
(d) Self 8 for Case 2.

Figure 6.13 Example of Selves for Failure Case 2: Right Aileron Stuck (Low Magnitude).

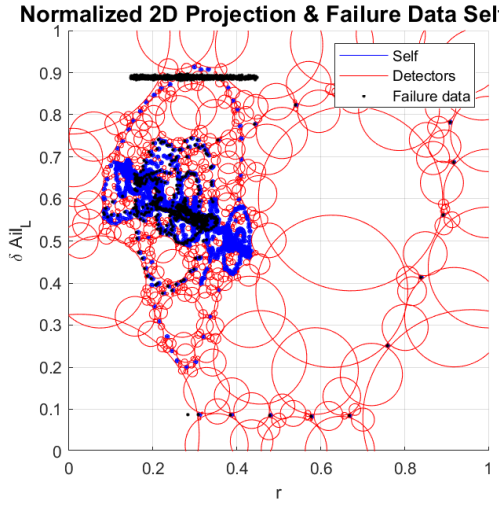
Figure 6.14 shows selves 2, 3, 5, and 8 for case 3. This case has increased DRs than the other cases and it is evident why: Self 2 in Figure 6.14a and Self 8 in Figure 6.14d show a great effort of the right aileron to try to compensate for the failure. Self 3 in Figure 6.14b and Self 5 in Figure 6.14c show again the constant line due to the left aileron locked at the fixed position. These selves are different from the previous cases, and this characteristic is exploited to identify the failure.



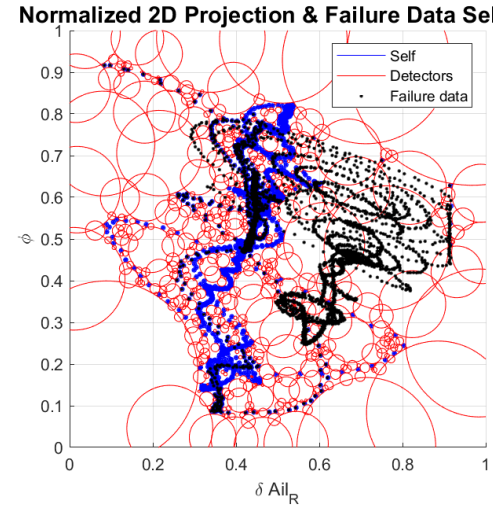
(a) Self 2 for Case 3.



(b) Self 3 for Case 3.



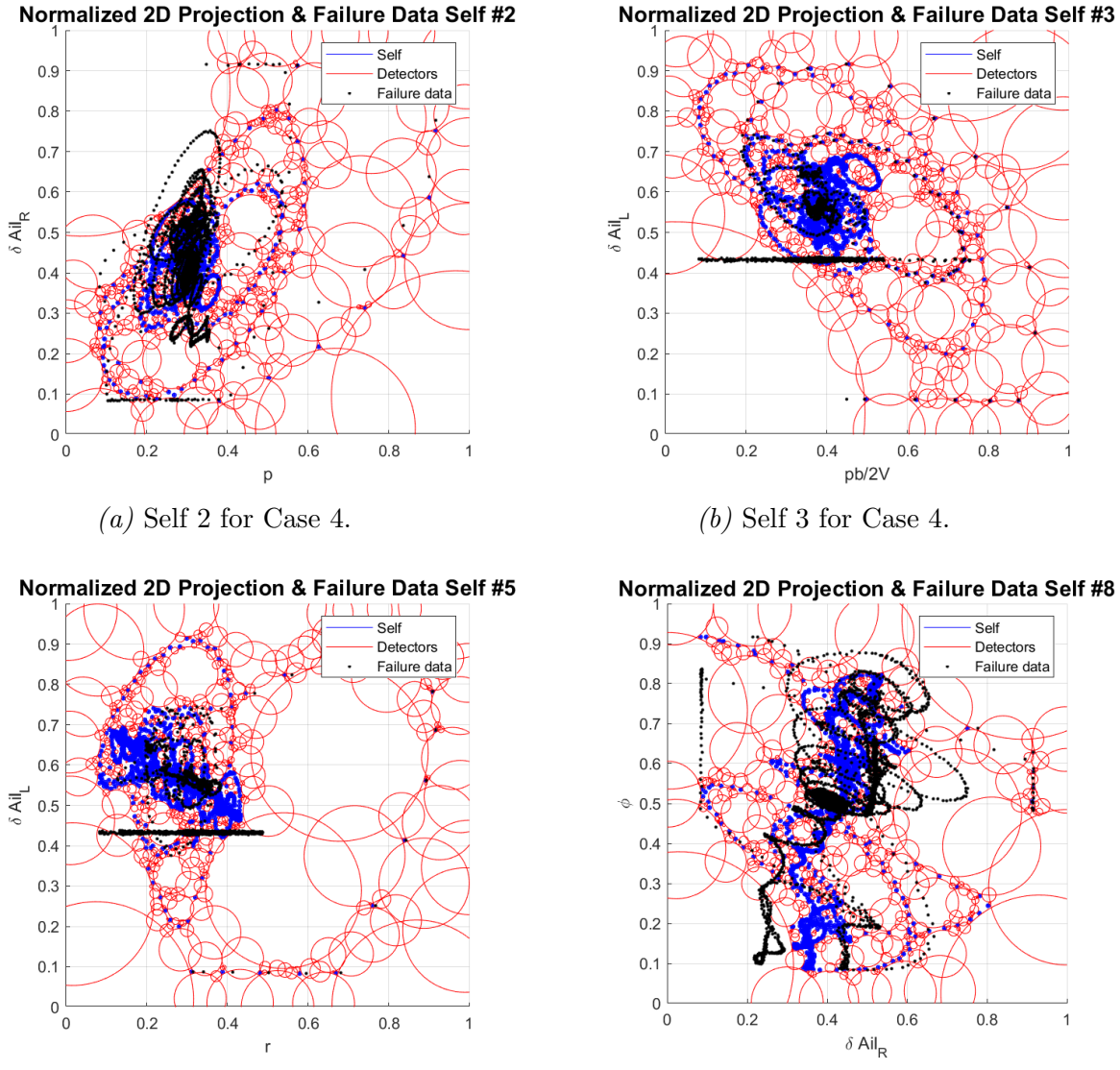
(c) Self 5 for Case 3.



(d) Self 8 for Case 3.

Figure 6.14 Example of Selves for Failure Case 3: Left Aileron Stuck (High Magnitude).

Finally, Figure 6.15 shows the same selected selves for case 4. The shape of the graphs is similar to the previous case, however the low magnitude failure makes the right aileron to make less effort. An important point to consider here is that there exists the possibility that some selves do not make any identification possible, but are only useful for detection. This is the case for self 8 in Figure 6.15d, where no clear conclusion can be made of whether the failure is on the left or right aileron. This is the reason why generating more selves will make the detection/identification phase more robust.



(a) Self 2 for Case 4.

(b) Self 3 for Case 4.

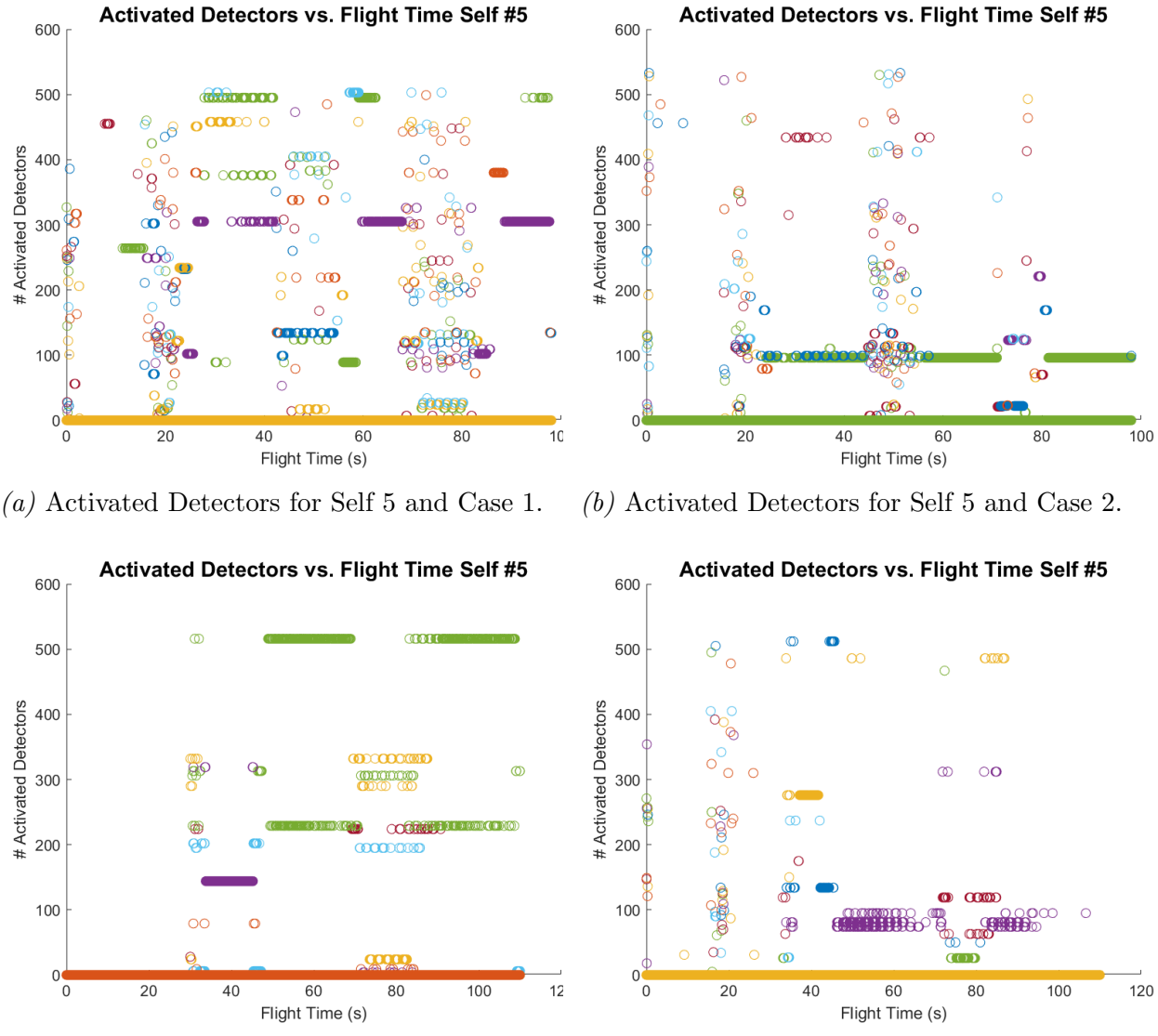
(c) Self 5 for Case 4.

(d) Self 8 for Case 4.

Figure 6.15 Example of Selves for Failure Case 4: Left Aileron Stuck (Low Magnitude).

6.4.2 Detection: Activated Detectors

The identification part is based on the specific activated detectors that range from 1 to 500. Depending on which detector is activated most of the time, a logic is developed such that a classification is done where a buffer of samples of the flight data is being recorded and compared online as described in section 4.3.4. Figure 6.16 shows an example of the time history of the NAD for self 5 in every failure case. Due to the effects of the AC, different antibodies are activated which allows the identification to take place correctly.

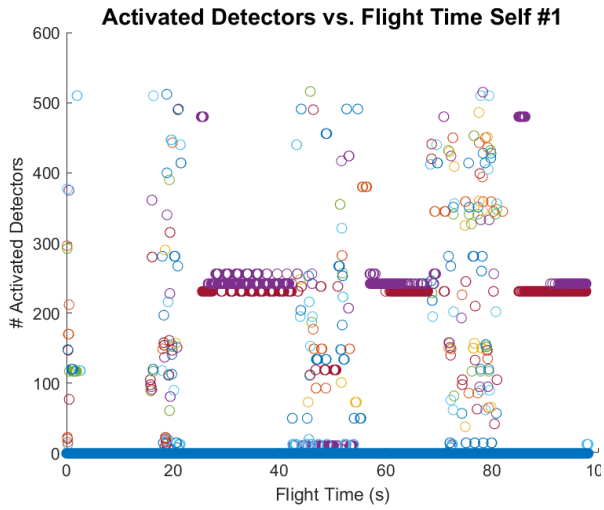


(a) Activated Detectors for Self 5 and Case 1. *(b)* Activated Detectors for Self 5 and Case 2.

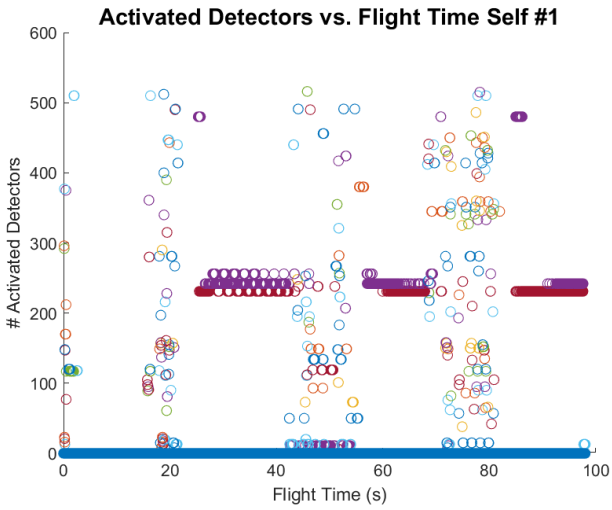
(c) Activated Detectors for Self 5 and Case 3. *(d)* Activated Detectors for Self 5 and Case 4.

Figure 6.16 Example of Activated Detectors for Self 5.

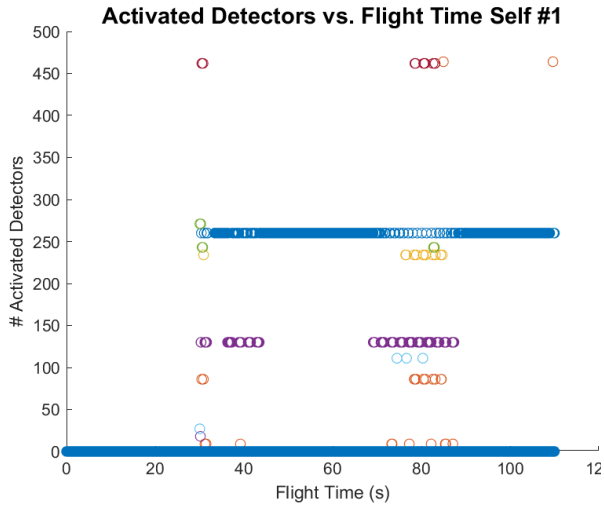
Similarly, Figure 6.17 highlights another example of NAD. This case is for self 1 that shows the projection for the roll rate and the left aileron deflection. Notice that Figure 6.17c and Figure 6.17d show more the activation of a specific detector in contrast with Figure 6.17a and Figure 6.17b since failure cases 3 and 4 are the ones that fail the left aileron.



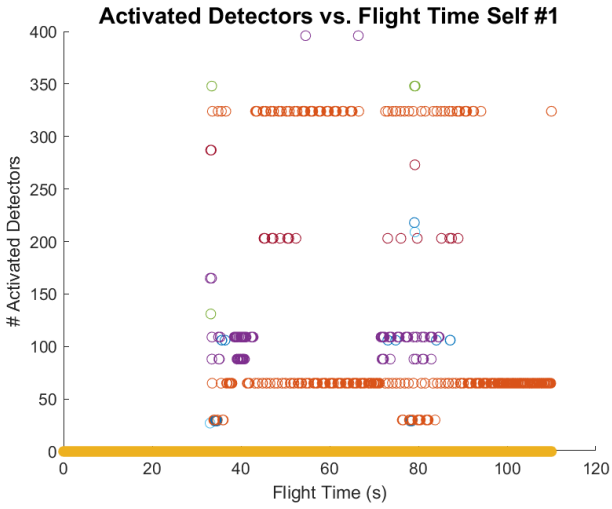
(a) Activated Detectors for Self 1 and Case 1.



(b) Activated Detectors for Self 1 and Case 2.



(c) Activated Detectors for Self 1 and Case 3.



(d) Activated Detectors for Self 1 and Case 4.

Figure 6.17 Example of Activated Detectors for Self 1.

After plotting the time history of all the NADs and for all the failure cases, the identification phase is complete and the closed-loop system containing the trained HMS-AIS can be integrated into the simulation. Figure 6.18 shows the complete simulation architecture in

action, where the failure from case 1 was inserted and detected. As pictured, the right aileron is stuck downwards and the left aileron is trying to compensate the failure after the replanning mission has been activated. The alarm message is constantly showing that a failure has been detected until the vehicle reaches its goal position.

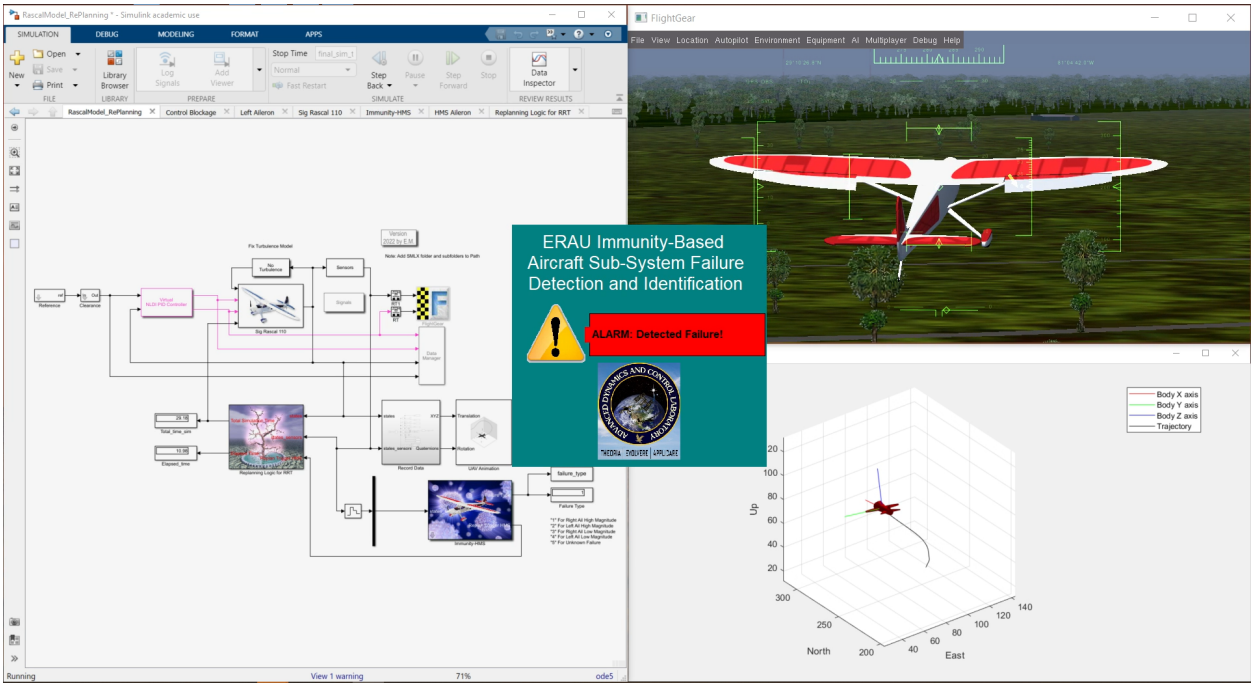


Figure 6.18 Failure Detection and Identification with Online HMS.

A final point that has to be addressed is the fact that the detection of the HMS can present some lag that depends on the window of samples selected. If a window of a high number of samples is chosen, the detection is delayed for more than 5 seconds, though the identification part becomes must robust. However, in order to maintain a balance between detection and identification, just a window of 10 samples was selected in such a way that the detection has a negligible delay of 0.2s and the identification phase still remains robust enough to activate the correct replanning mission.

7 Conclusions & Future Work

A HMS based on the AIS paradigm has been successfully implemented in junction with a multi-goal RRT* path planner in order to generate safe trajectories for a fixed-wing UAV under reduced flight envelope conditions due to aileron failures. The concept contributes to the idea of having on-board artificial intelligence based on an autonomous decision-making architecture to prevent the UAV to present potential risks of damage, collision or harm to private/public property and people in case of an AC during flight. Additionally, this contribution helps to advance one step further the safely integration of these type of unmanned vehicles to the NAS.

The use of 3D occupancy maps, optimized to be memory-efficient, is used to build the 3D environment that simulates a typical urban area. The map is made up by using unit voxels that determine whether the free space is occupied or not. Furthermore, it is assumed that the map is already known, although the approach is still valid in case other types of obstacle detection methods, such as vision system-based approaches, are implemented.

An analysis of the efficiency in terms of cost functions, computational effort, and trajectory smoothness has been carried out in order to choose the RRT* as the main path planner for this approach. The original RRT* planner is combined with the Dubins Airplane Paths to take into account the kinodynamic properties of the vehicle within the 3D map, and at the same time to smooth the trajectory according to restrictions in roll capabilities of the aircraft. Additionally, decision-making properties have been integrated into the planner to generate feasible trajectories within the vehicle's physical limitations.

The HMS has been developed under the AIS paradigm with the goal of detecting and identifying specific type of failures related to the jammed ailerons at fixed positions. A closed-loop architecture is then proposed to merge the path planner and then HMS to increase the autonomy of the replanned missions. With this, a simulation environment containing the dynamic model of the Rascal 110 has been interfaced with FlightGear and Simulink to test the generation and feasibility of the new save trajectories.

Further improvements can enhance several aspects of the algorithms used in this study. First, notice that the HMS still must undergo a supervised learning process to capture the dynamic fingerprint of the aircraft, and the selves must be manually trained to accurately identify the detected failures. Ideally, the HMS should be able to "adapt" to different flight conditions and the detection/identification phases would be able to trigger alarms with different types of subsystem failures, and not only for control surface failures. This adaptation can be implemented and improved by using other methodologies such as Support Vector Machine (SVM).

For a real-world application, the 3D occupancy maps can be built based on vision system algorithms such that a preloaded map would not be needed to implement the architecture. This approach is feasible to work online and on an on-board computer due to the memory-efficient property of these types of maps.

At last, the Airplane Dubins Paths could be further improved to integrate more restrictions in the flight envelope that could come from failures in other subsystems, such as the propulsion and electrical segments of the aircraft.

These implementations and further improvements can make the entire architecture feasible to be applied not only for fixed-wings, but for any type of aerial vehicle such as quadcopters, octocopters, and VTOLs.

REFERENCES

[1] Federal Aviation Administration, United Sates Of America, "FAA Aerospace Forecast Fiscal Years 2020-2040." , 2020. https://www.faa.gov/data_research/aviation/aerospace_forecasts/media/FY2020-40_FAA_Aerospace_Forecast.pdf, Last accessed on May 10, 2022.

[2] Namian, M., Khalid, M., Wang, G., and Turkan, Y., "Revealing Safety Risks of Unmanned Aerial Vehicles in Construction," *Journal of the Transportation Research Board*, Vol. 2675, 2021, pp. 334 – 347. https://doi.org/10.1177/03611981211017134.

[3] Jeong, S., You, K., and Seok, D., "Hazardous flight region prediction for a small UAV operated in an urban area using a deep neural network," *Aerospace Science and Technology*, Vol. 118, 2021, p. 107060. https://doi.org/10.1016/j.ast.2021.107060.

[4] Bakori, M., "UAS Model Identification and Simulation to Support In-Flight Testing of Discrete Adaptive Fault-Tolerant Control Laws," Master's thesis, Embry-Riddle Aeronautical University, 2020.

[5] Asadi, K., Kalkunte Suresh, A., Ender, S., Gotad, S., Maniyar, S., Noghabaei, K., Lobaton, E., and Wu, T., "An Integrated UGV-UAV System for Construction Site Data Collection." *Automation in Construction*, Vol. 112, 2020, p. 103068. https://doi.org/https://doi.org/10.1016/j.autcon.2019.103068.

[6] Valavanis, K. P., and Vachtsevanos, G. J. (eds.), *Handbook of Unmanned Aerial Vehicles*, Springer Dordrecht, 2015. https://doi.org/10.1007/978-90-481-9707-1.

[7] Beard, W. R., and McLain, E. T., *Small Unmanned Aircraft: Theory and Practice*, Princeton University Press, 2012.

[8] Moncayo, H., Perhinschi, M. G., and David, J., "Aircraft Failure Detection and Identification Using an Immunological Hierarchical Multiself Strategy," *Journal of Guidance, Control, and Dynamics*, Vol. 33, 2010, pp. 1105 – 1114. https://doi.org/10.2514/1.47445.

[9] Newman, P., Cole, D., and Ho, K., "Outdoor SLAM using visual appearance and laser ranging," *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.*, 2006, pp. 1180–1187. https://doi.org/10.1109/ROBOT.2006.1641869.

[10] Douillard, B., Underwood, J., Melkumyan, N., Singh, S., Vasudevan, S., Brunner, C., and Quadros, A., "Hybrid elevation maps: 3D surface models for segmentation," *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2010, pp. 1532–1538. https://doi.org/10.1109/IROS.2010.5650541.

[11] Triebel, R., Pfaff, P., and Burgard, W., "Multi-Level Surface Maps for Outdoor Terrain Mapping and Loop Closing," *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2006, pp. 2276–2282. https://doi.org/10.1109/IROS.2006.282632.

[12] Ryde, J., and Hu, H., "3D mapping with multi-resolution occupied voxel lists," *Autonomous Robots*, Vol. 28, No. 2, 2009, p. 169–185. https://doi.org/10.1007/s10514-009-9158-3.

[13] Wurm, K. M., Hornung, A., Bennewitz, M., Stachniss, C., and Burgard, W., "OctoMap: A probabilistic, flexible, and compact 3D map representation for robotic systems," *In Proc. of the ICRA 2010 workshop*, 2010.

[14] Hornung, A., Wurm, M. K., Bennewitz, M., Stachniss, C., and Burgard, W., "OctoMap: an efficient probabilistic 3D mapping framework based on octrees," *Auton Robot*, Vol. 34, 2013, pp. 190 – 206. https://doi.org/10.1007/S10514-012-9321-0.

[15] Karur, K., Sharma, N., Dharmatti, C., and Siegel, J. E., "A survey of path planning algorithms for Mobile Robots," *Vehicles*, Vol. 3, No. 3, 2021, p. 448–468. https://doi.org/10.3390/vehicles3030027.

[16] Yang, L., Qi, J., Song, D., Xiao, J., Han, J., and Xia, Y., "Survey of robot 3D path planning algorithms," *Journal of Control Science and Engineering*, Vol. 2016, 2016, p. 1–22. https://doi.org/10.1155/2016/7426913.

[17] LaValle, S. M., "Rapidly-exploring random trees : a new tool for path planning," *The annual research report*, 1998.

[18] Wilburn, J. N., "Development of an Integrated Intelligent Multi -Objective Framework for UAV Trajectory Generation," Ph.D. thesis, West Virginia University, 2013.

[19] Dijkstra, E. W., "A note on two problems in connection with graphs," *Numerische Mathematik*, Vol. 1, No. 1, 1959, p. 269–271. https://doi.org/10.1007/bf01386390.

[20] Koenig, S., and Likhachev, M., "Improved fast replanning for robot navigation in unknown terrain," *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No.02CH37292)*, Vol. 1, 2002, pp. 968–975 vol.1. https://doi.org/10.1109/ROBOT.2002.1013481.

[21] Koenig, S., and Likhachev, M., "Fast replanning for navigation in unknown terrain," *Robotics, IEEE Transactions on*, Vol. 21, 2005, pp. 354 – 363. https://doi.org/10.1109/TRO.2004.838026.

[22] Yue, R., Xiao, J., Wang, S., and Joseph, S. L., "Modeling and Path Planning of the City-Climber Robot Part II: 3D Path Planning Using Mixed Integer Linear Programming," IEEE Press, 2009, p. 2391–2396.

[23] Masehian, E., and Habibi, G., "Robot Path Planning in 3D Space Using Binary Integer Programming," Vol. 1, 2007.

[24] Chamseddine, A., Zhang, Y., Rabbath, C. A., Join, C., and Theilliol, D., "Flatness-Based Trajectory Planning/Replanning for a Quadrotor Unmanned Aerial Vehicle," *IEEE Transactions on Aerospace and Electronic Systems*, Vol. 48, No. 4, 2012, pp. 2832–2848. https://doi.org/10.1109/TAES.2012.6324664.

[25] Dorigo, M., Maniezzo, V., and Colorni, A., "Ant system: optimization by a colony of

cooperating agents," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, Vol. 26, No. 1, 1996, pp. 29–41. https://doi.org/10.1109/3477.484436.

[26] Yang, X.-S., *A New Metaheuristic Bat-Inspired Algorithm*, chapter and pages, pp. 65–74. https://doi.org/10.1007/978-3-642-12538-6_6, URL https://doi.org/10.1007/978-3-642-12538-6_6.

[27] Holland, J. H., *Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and Artificial Intelligence*, MIT Press, 2010.

[28] Lin, N., Tang, J., Li, X., and Zhao, L., "A novel improved bat algorithm in UAV path planning," *Computers, Materials amp; Continua*, Vol. 61, No. 1, 2019, p. 323–344. https://doi.org/10.32604/cmc.2019.05674.

[29] Modin, K., and Verdier, O., "What makes nonholonomic integrators work?" *Numerische Mathematik*, Vol. 145, No. 2, 2020, p. 405–435. https://doi.org/10.1007/s00211-020-01126-y.

[30] Divelbiss, A., Seereeram, S., and Wen, J. T., *Kinematic Path Planning for Robots with Holonomic and Nonholonomic Constraints*, chapter and pages, pp. 127–150. https://doi.org/10.1007/978-1-4612-1710-7_5, URL https://doi.org/10.1007/978-1-4612-1710-7_5.

[31] Karaman, S., and Frazzoli, E., "Sampling-based Algorithms for Optimal Motion Planning," *Auton Robot*, 2011, pp. 1 – 76. https://doi.org/10.48550/arXiv.1105.1186.

[32] Gammell, J. D., and Strub, M. P., "Asymptotically Optimal Sampling-Based Motion Planning Methods," *Annual Review of Control, Robotics, and Autonomous Systems*, Vol. 4, No. 1, 2021, pp. 295–318. https://doi.org/10.1146/annurev-control-061920-093753, URL https://doi.org/10.1146%2Fannurev-control-061920-093753.

[33] Dubins, L., "On Curves of Minimal Length with a Constraint on Average Curvature,

and with Prescribed Initial and Terminal Positions and Tangents," *The Johns Hopkins University Press*, Vol. 79, 1957, pp. 497–516.

[34] Dasgupta, D., and Nino, F., *Immunological Computation: Theory and Applications*, Taylor & Francis, 2008. URL https://doi.org/10.1201/9781420065466.

[35] Takahashi, K., and Yamada, T., "Application of an immune feedback mechanism to control systems." *JSME International Journal Series C*, Vol. 41, No. 2, 1998, p. 184–191. https://doi.org/10.1299/jsmec.41.184.

[36] Moncayo, H., Perhinschi, M. G., and Davis, J., "Artificial-Immune-System-Based Aircraft Failure Evaluation over Extended Flight Envelope," *Journal of Guidance, Control, and Dynamics*, Vol. 34, 2022, pp. 989 – 1001. https://doi.org/10.2514/1.52748.

[37] Kaneshige, J., and Krishnakumar, K., "Artificial immune system approach for air combat maneuvering," *Intelligent Computing: Theory and Applications V*, 2007, pp. 989 – 1001. https://doi.org/10.1117/12.718892.

[38] "The role of B cells and T cells in COVID-19 immune response &; T cell immunity in COVID-19 patients," , May 2022. URL https://www.akadeum.com/blog/an-inside-look-at-the-role-of-t-cells-and-b-cells-in-immune-response-to-covid-19/.

[39] Sansonetti, P., and Zychlinsky, A., *Molecular Cellular Microbiology*, Methods in microbiology, Vol. 31, Academic Press, 2002.

[40] Janeway, C. A., Travers, P., Walport, M., and Capra, D. J., *Immunobiology*, Taylor & Francis Group UK: Garland Science, 2001.

[41] Perhinschi, M. G., Al Azzawi, D., Moncayo, H., Perez, A., and Togayev, A., "Immunity-based aircraft actuator failure evaluation," *Aircraft Engineering and Aerospace Technology*, Vol. 88, 2016, pp. 729 – 739. https://doi.org/10.1108/AEAT-07-2014-0117.

[42] Perhinschi, M. G., Al Azzawi, D., Moncayo, H., Perez, A., and Togayev, A., "Immunity-based flight envelope prediction at post-failure conditions," *Aircraft Engineering and Aerospace Technology*, Vol. 46, 2015, pp. 264 – 272. https://doi.org/10.1016/j.ast.2015.07.014.

[43] Garcia, D., "Design, Development and Implementation of Intelligent Algorithms to Increase Autonomy of Quadrotor Unmanned Missions," Master's thesis, Embry-Riddle Aeronautical University, 2017.

[44] La Valle, S. M., *Planning Algorithms*, Cambridge University Press, 2006.

[45] Chitsaz, H., and LaValle, S. M., "Time-optimal paths for a Dubins airplane," *2007 46th IEEE Conference on Decision and Control*, 2007, pp. 2379–2384. https://doi.org/10.1109/CDC.2007.4434966.

[46] Owen, M., Beard, R. W., and McLain, T. W., "Implementing Dubins Airplane Paths on Fixed-Wing UAVs," *Handbook of Unmanned Aerial Vehicles*, edited by K. P. Valavanis and G. J. Vachtsevanos, Springer Dordrecht, 2015, Chap. 68, p. 1691–1697.

[47] Ducard, G., Kulling, K. C., and Geering, H. P., "Evaluation of reduction in the performance of a small UAV after an aileron failure for an adaptive guidance system," *2007 American Control Conference*, 2007. https://doi.org/10.1109/acc.2007.4282845.

[48] Perhinschi, M. G., and Moncayo, H., *Artificial Immune System for Comprehensive and Integrated Aircraft Abnormal Conditions Management*, American Institute of Aeronautics and Astronautics Inc., 2018.

[49] Lyons, B., "Performance Analysis Of Non-Linear Adaptive Control Laws Using Hardware in the Loop of an Unmanned Aerial System," Master's thesis, Embry-Riddle Aeronautical University, 2013.

[50] O'Toole, S., "Development of a Remotely-Piloted Vehicle Platform to Support Implementation, Verification, and Validation of Pilot Control Systems," Master's thesis, Embry-Riddle Aeronautical University, 2017.

[51] Nelson, R., *Flight Stability and Automatic Control*, Aerospace Science & Technology, WCB/McGraw Hill, 1998. URL https://books.google.com.ec/books?id=Uzs8PgAACAAJ.

[52] Campa, G., Napolitano, M., Seanor, B., and Perhinschi, M., "Design of control laws for maneuvered formation flight," *Proceedings of the 2004 American Control Conference*, Vol. 3, 2004, pp. 2344–2349 vol.3. https://doi.org/10.23919/ACC.2004.1383814.

[53] Moncayo, H., Perhinschi, M., Wilburn, B., Wilburn, J., and Karas, O., "Extended nonlinear dynamic inversion control laws for unmanned Air Vehicles," *AIAA Guidance, Navigation, and Control Conference*, 2012. https://doi.org/10.2514/6.2012-4675.

[54] Verberne, J., and Moncayo, H., "Comparison of Adaptive Control Laws for Wind Rejection in Quadrotor UAVs," 2019. https://doi.org/10.1115/DSCC2019-8957, URL https://doi.org/10.1115/DSCC2019-8957.